Data Management • Enterprise Solutions • XML Labs

XML JOURNAL

THE ULTIMATE XML ENTERPRISE RESOURCE

November 2003 Volume:4 Issue:11

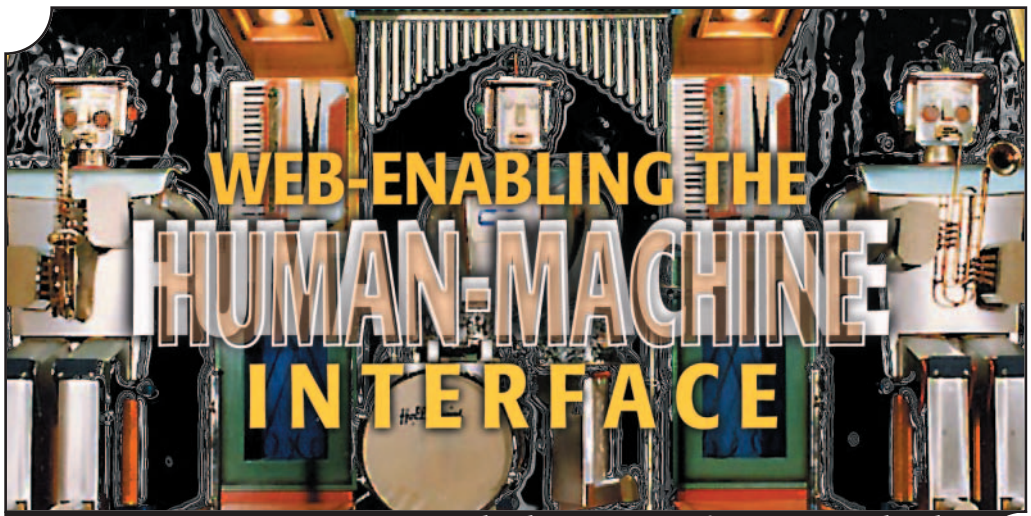
DISPLAY UNTIL January 31, 2004

$6.99US $7.99CAN

12

0 09281 01314 3

SYS-CON MEDIA



SVG extends the power of open standards 18

XML JOURNAL

FOUNDING EDITOR
Ajit Sagar ajit@sys-con.com

EDITORIAL ADVISORY BOARD
Graham Glass graham@themindelectric.com
Coco Jaenicke cjaenicke@attbi.com
Sean McGrath sean.mcgrath@propylon.com
Simeon Simeonov talktosim@polarisventures.com

EDITORIAL
Editor-in-Chief
Hitesh Seth hitesh@sys-con.com
Managing Editor
Jennifer Van Winckel jennifer@sys-con.com
Editor
Nancy Valentine nancy@sys-con.com
Associate Editors
John Evdemon jevdemon@sys-con.com
Jamie Matusow jamie@sys-con.com
Gail Schultz gail@sys-con.com
Jean Cassidy jean@sys-con.com

PRODUCTION
Production Consultant
Jim Morgan jim@sys-con.com
Art Director
Alex Botero alex@sys-con.com
Associate Art Directors
Louis F. Cuffari louis@sys-con.com
Richard Silverberg richards@sys-con.com
Assistant Art Director
Tami Beatty tami@sys-con.com

CONTRIBUTORS TO THIS ISSUE
PG Bartlett, Enrique Castro, Michael M David, Jeremy Geelan, Eugene Kuznetsov, Michael Leventhal, Doron Sherman, Steve Vinoski, Greg Watson, Rob Williamson

EDITORIAL OFFICES
SYS-CON MEDIA
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9637
XML-JOURNAL (ISSN# 1534-9780)
is published monthly (12 times a year)
by SYS-CON Publications, Inc.
Periodicals postage pending
Montvale, NJ 07645 and additional mailing offices.
POSTMASTER: Send address changes to:
XML-JOURNAL, SYS-CON Publications, Inc.,
135 Chestnut Ridge Road, Montvale, NJ 07645.
Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ

FOR LIST RENTAL INFORMATION:
Kevin Collopy: 845 731-2684,
kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832,
frank.cipolla@epostdirect.com

SYS-CON MEDIA

Multi-Middleware Web Services

WRITTEN BY **STEVE VINOSKI**

A few years ago, integrating applications simply required putting the right middleware in place. Today, however, the proliferation of different middleware approaches and technologies has complicated matters. Application integration now implies middleware integration. Essentially, we need "middleware for middleware" that allows us to noninvasively integrate disparate middleware systems.

The economic downturn of recent years has led many to question large and expensive "rip and replace" projects. Such projects, which attempt to homogenize the enterprise by ripping everything out and replacing it with the latest silver bullet, almost always fail. IT groups instead prefer to leave working systems in place and get them to talk to each other without changing them, regardless of which suppliers they come from, which technologies they employ, what languages they're written in, or what protocols they speak.

Middleware integration requires finding abstractions rich enough to represent a wide variety of design approaches, and yet concrete enough to embrace and support the numerous technologies used for service implementation. Naturally, finding practical middleware integration approaches is not easy. For example, early Web services adopters embraced SOAP because it was much easier to add to new systems than other traditional middleware. However, SOAP is only a network protocol, and thus it can't provide all the abstractions necessary for multi-middleware integration. In fact, SOAP does nothing to help link together systems that are already deployed, because adding SOAP to such applications is just as invasive as previous middleware approaches. Invasive changes to working systems are error-prone, and they result in expensive and time-consuming retesting and redeployment.

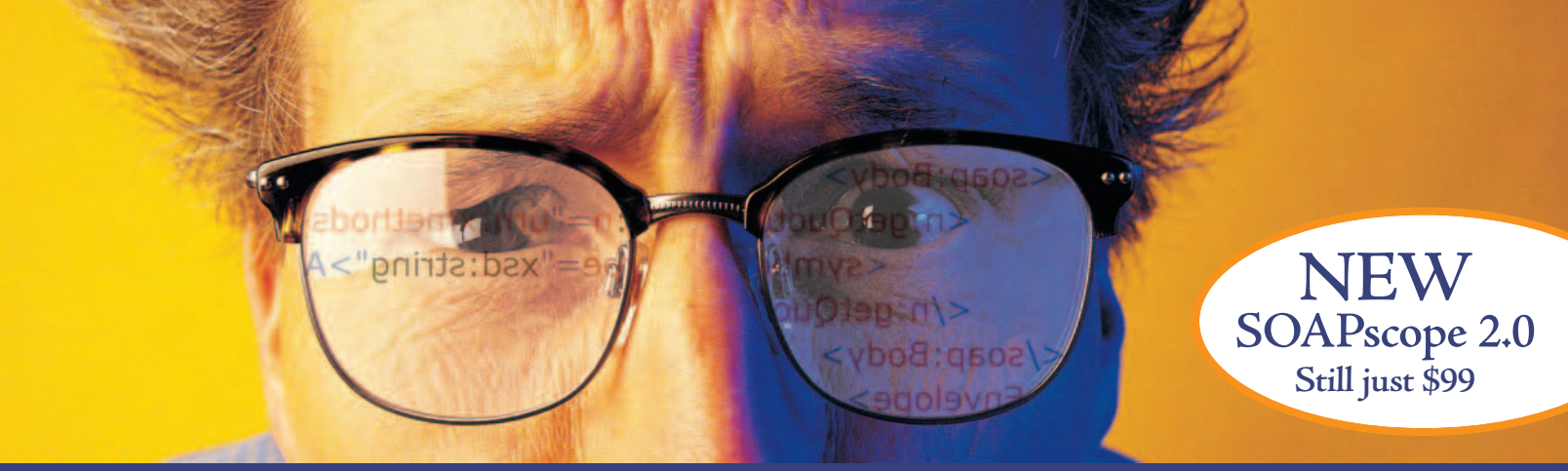
In an ideal world, multi-middleware integration would allow you to create new business services by getting your existing systems to interoperate without changing them at all. But isn't this what Enterprise Application Integration (EAI) is all about? EAI proponents have long claimed that it would supply easy, seamless, and noninvasive integration, but in practice, this has usually not been the case. Typical EAI projects cost millions of dollars, run well behind schedule, and ultimately fail to deliver viable integrated systems.

Fortunately, Web services technologies provide exactly the abstractions required for noninvasive multi-middleware integration. Specifically, the features and flexibility of the Web Services Description Language (WSDL) enable you to "reverse engineer" existing applications to prepare them for integration with other disparate systems. WSDL types and port/interface definitions support logical descriptions of existing services, while WSDL binding extensions define physical connections to those services not in terms of SOAP, but in terms of their existing protocols and message formats. Combined with the right underlying "middleware for middleware" support, WSDL-based abstractions and bindings can be used to create integrated systems that employ multiple protocols, multiple message formats, and multiple interconnection patterns.

One project using WSDL in this fashion is the Apache Web Services Invocation Framework (WSIF). Because SOAP alone is not enough for real-world integration, the goal of WSIF is to supply a Java API that allows an application to transparently invoke Web services via a variety of protocols. With this approach, service contracts can be defined abstractly using the logical features of WSDL. Then, for each service instance, these logical definitions can be joined with the appropriate physical bindings that define connectivity to that service. The logical/physical separation allows client applications to depend only on the logical portion of the WSDL definition, keeping them independent of the actual protocols and message formats required to communicate with a specific service instance. Client applications rely on separate WSIF providers that support connecting to services implemented using EJB, JMS, JCA, or local Java objects. These providers can be dynamically loaded, allowing the actual binding to a service to be chosen at runtime. Adding support for services implemented using another middleware system, such as MQ Series, simply requires writing a new provider. WSIF makes these different provider implementations completely transparent to client applications.

The WSIF approach is definitely a step above the typical non-abstracted, SOAP-only Web service applications that other frameworks support. Ultimately, though, multi-middleware integration requires features beyond those of WSIF. In fact, multiple approaches are

~continued on page 21~



NEW
SOAPscope 2.0
Still just $99

4 ways to...KNOW YOUR WSDL

No matter how robust the system, one small WSDL mistake can break everything. That's why Mindreef integrated 4 powerful WSDL diagnostics into SOAPscope. SOAPscope 2.0 makes it easy to keep your WSDL clean.

See it: To be productive with WSDL you need different views for different tasks. Only SOAPscope gives you pseudocode, tree, syntax-colored XML & raw views to quickly understand **any** WSDL.

Try it: Invoke Web services without writing any code. "Test drive" a web service through an easy-to-use form. Easily trying different conditions is essential for productive troubleshooting, testing or debugging.

Diff it: A small change in a WSDL can cause a hard-to-find problem. Diff pinpoints the change and is fully integrated with the other SOAPscope diagnostics so it's there when you need it.

Check it: Check for validity, compliance and conformance against the W3C WSDL 1.1 standard, the WS-I Basic Profile and Mindreef's best practices learned in the Web Services trenches. If your WSDL doesn't pass cleanly through SOAPscope, it's not fully interoperable!

SOAPscope combines the industry's best Web services diagnostic tools into a powerful, comprehensive, easy-to-use package. SOAPscope is the only Web services diagnostic tool you will need whether you're creating your first WSDL or debugging production services.

" Finally there's a product that makes it easy for developers to tackle WSDL: SOAPscope 2.0! "

Aaron Skonnard,
Instructor, DevelopMentor.

Let Mr. SOAPscope check YOUR WSDL!

FREE evaluation at www.mindreef.com

"The Web Services Diagnostic Experts" Mindreef

Rise of the Standards-Based Integration Machines

WRITTEN BY **DORON SHERMAN**

It occurs to me that my choice of title for this guest editorial may be at least partially influenced by the recall-induced elections in California (can you see the Arnie connection?). But this column is not about politics; it's about a new, industry-standard ecosystem built around XML to address today's business integration and process automation challenges.

The nature of technology ecosystems is that they are created piecemeal, usually from the bottom up. Following this model, XML initially provided a common syntax for capturing and expressing data within documents. Next, XML Schema emerged to serve as a template for describing and enforcing document structures. Finally, functional entities, known as Web services, became standardized using XML to describe their operations (WSDL) and to exchange data over Internet transport layers (via SOAP, an XML-based protocol).

As the basic standards stack has evolved to support heterogeneous resources and data across the network, the time has come for the next layer, which addresses enterprise integration problems and increases business agility and flexibility. Once XML resources are published, consuming them, aggregating them, and coordinating control flow among them is the natural next step. This drove the emergence of BPEL4WS (aka BPEL), which provides an XML-based orchestration language for executing business processes.

Several requirements remain. Processing and transforming XML documents and basic algebra functionality, such as comparing and adding values together, are needed. In BPEL, these expressions are carried out using XPath. And since XML provides a platform-independent mechanism of integrating disparate systems, a flexible XML transformation capability can be effectively addressed with the use of XQuery.

Have you caught an important piece that I have missed so far? The preceding text portrays IT assets and resources as readily available via XML Web services protocols – hardly the case today. To our rescue comes a novel concept that I call the binding framework. Rather than migrating existing IT assets into XML and SOAP services (often uneconomical or inefficient), such assets are described by a WSDL file that defines the interface in a standardized way. The versatile binding framework then selects the appropriate protocol for communicating with the IT asset, based on its interface file.

To help handle the impending complexity and a potential for brittle and change-resistant integration architectures, is a concept called SOA, or service-oriented architecture. XML lends itself to building modular, loosely coupled systems. The standards-based integration machine comprises a set of built-in services, all working in tandem to provide the functions needed for enterprise integration and process automation. At the heart of such a machine lies the engine for orchestrating BPEL processes that utilize such services as document transformation, legacy systems access, and business rules execution.

One of the primary drivers for the emergence of standards-based integration machines is the rapidly growing BPM (business process management) market. In its market milestone report on BPM, Delphi Group touts BPM as the "final great boom of the software industry" and indicates a $550 million market for BPM in 2003 with 15%–30% growth expected for the next three years. A notable focus is on orchestration of enterprise-wide processes in which BPM constitutes an emerging layer of software for building new process-based applications.

These trends are not lost on enterprise software vendors with backgrounds in EAI, B2B, BPM, workflow, and Web services technologies as well as incumbent platform vendors who see the need to expand application development into the integration domain. The XML ecosystem is driving convergence among the seemingly disparate software categories to adopt SOA and the new standards stack. Whether called ESB (enterprise services bus) or BPEL orchestration server, the rise of the standards-based integration machines is a sign of material changes to the traditional integration landscape. The JSR 208 extension (Java Business Integration) of the J2EE platform captures the essence of this new architecture.

The days of having to choose from expensive proprietary solutions, risky build-your-own infrastructure, and do-nothing-and-wait approaches are over. XML powers a simple, flexible approach that features rich process semantics, enhanced execution visibility, continuous adaptation, and optimization of business processes, and reduces vendor lock-in.

Author Bio

Doron Sherman is the CTO of Collaxa, Inc., a Web Service Orchestration Server vendor and a BEA Partner located in Redwood Shores, California. He has been involved with Java since its early days and pioneered application server technology while being a founder and chief scientist at NetDynamics.

DORON@COLLAXA.COM

SYS-CON MEDIA

135 Chestnut Ridge Rd., Montvale, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9637

PRESIDENT and CEO
Fuat Kircaali fuat@sys-con.com

VP, Business Development
Grisha Davida grisha@sys-con.com
Group Publisher
Jeremy Geelan jeremy@sys-con.com
Technical Director
Alan Williamson alan@sys-con.com

ADVERTISING
Senior VP, Sales & Marketing
Carmen Gonzalez carmen@sys-con.com
VP, Sales & Marketing
Miles Silverman miles@sys-con.com
Advertising Director
Robyn Forma robyn@sys-con.com
Director of Sales & Marketing
Megan Mussa megan@sys-con.com
Advertising Sales Manager
Alisa Catalano alisa@sys-con.com
Associate Sales Managers
Carrie Gebert carrieg@sys-con.com
Kristin Kuhnle kristin@sys-con.com

SYS-CON EVENTS
President
Grisha Davida grisha@sys-con.com
Conference Manager
Michael Lynch mike@sys-con.com
National Sales Manager
Sean Raman raman@sys-con.com

CUSTOMER RELATIONS
Circulation Service Coordinators
Niki Panagopoulos niki@sys-con.com
Shelia Dickerson shelia@sys-con.com
Edna Earle Russell edna@sys-con.com

JDJ STORE
Manager
Rachel McGouran rachel@sys-con.com

WEB SERVICES
VP, Information Systems
Robert Diamond robert@sys-con.com
Web Designers
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
Online Editor
Lin Goetz lin@sys-con.com

ACCOUNTING
Accounts Receivable
Kerri Von Achen kerri@sys-con.com
Financial Analyst
Joan LaRose joan@sys-con.com
Accounts Payable
Betty White betty@sys-con.com

SUBSCRIPTIONS
SUBSCRIBE@SYS-CON.COM
1 888 303-5282
For subscriptions and requests for bulk orders, please send your letters to Subscription Department
Cover Price: $6.99/issue
Domestic: $69.99/yr (12 issues)
Canada/Mexico: $89.99/yr
all other countries $99.99/yr
(U.S. Banks or Money Orders)
Back issues: $12 U.S. $15 all others

WRITTEN BY **EUGENE KUZNETSOV**

XML in Unexpected Places

Understanding the emergence of XML outside software systems

It is a welcome sign of XML's success that it increasingly appears in unexpected places and concerns an ever-broadening set of people. Those of you who were involved with XML technology in the early days or who bravely pioneered XML in your organizations years ago may be bewildered by this, and wonder whether all of these "other" people really understand what XML is all about. The major change is that for the business people, the question has now irrevocably become "When XML?" not "XML?". Plugging our ears to keep out the din of marketing noise, let's take a quick tour of the unexpected places you'll find XML.

XML in Firewalls

It is quite natural to ask "Huh? What does XML have to do with firewalls?". XML is quickly becoming part of the network security infrastructure for three major reasons. First, XML Web services were designed to bypass the existing firewalls and intrusion detection systems. You may remember early articles about SOAP that promoted it as "firewall-friendly RPC for Windows." The last thing any security professional wants, especially in the aftermath of MSBlaster, is "firewall-friendly" anything for Windows. Detecting that a particular HTTP connection is actually carrying XML/SOAP and ensuring that the SOAP requests are valid requires an XML-aware firewall, leading a number of startup vendors to develop software and hardware products for the job.

Second, there is a larger trend of moving application-level security into network devices, motivated in part by the realization that most security threats today are at the application layer, and keeping up with the constant flow of application patches turns out to be very difficult in practice. By establishing a universal data encoding for application data and some basic security building blocks, XML makes the move to message-level security easier and more attractive. In addition to securing the enterprise perimeter or encrypting individual connections (using SSL or VPN), you can encrypt and tamper-proof individual messages – regardless of whether they are in transit, stored on disk, or transported hop-by-hop in a multiparty business workflow.

> "The day is not far off when all of your personal communications will be unexpectedly controlled by XML"

The third and final reason for the appearance of XML in firewalls is the need for separation of concerns between network operations, security, and application development groups in large corporations. Applications are usually deployed and managed not by the developers who build them, but by operations people. Even if it were easy to fully secure Web services traffic inside application code, the security group would still need to be able to change access policies, monitor for breaches, and manage the lower portions of the security stack. This usually does not work well because there is no clean separation of security functions from the application business logic, deploying the same policy to many servers is time-consuming (especially for those used to managing network appliances), and inevitable battles over policy gray areas arise. As a result, moving corporate-wide security policy enforcement to a network device managed by the security group is more scalable and provides some additional checks and balances.

In short, SOAP is "firewall friendly," which is a euphemism for the ability to slice through firewalls like a hot knife through butter – and so there's a need to secure it. To secure it intelligently, a firewall or gateway must be XML-aware. This plays into a larger trend of shifting from transport-layer to message-level security. People dynamics, security hygiene, and organizational politics drive separation of concerns and moving this security into dedicated devices. The whole trend is an opportunity to finally create a universal application security layer (see Figure 1).

Personal XML

In a departure from its document-publishing and invoice-schema roots, XML is also used in IM. The most prominent is probably Jabber and its XML-based protocol that is now beginning to be adopted for other, non-IM uses. XML, sometimes in its binary-encoded forms, has long been present in cellphones and PDAs, and it is starting to gather momentum as a control and configuration protocol in telephone networks. The day is not far off when all of your personal communications – from Instant Messaging, to PDA, to cellphone and land line – will be unexpectedly controlled by XML!

AUTHOR BIO

Eugene Kuznetsov, chairman and CTO, founded DataPower Technology, Inc., in 1999 to provide enterprises with intelligent XML-Aware Network (XAN) infrastructure to ensure unparalleled security, performance, and manageability for XML and Web services applications. He served as president of the company until spring 2003. Eugene is a technology visionary who has worked on real-world enterprise XML issues since the late '90s.

Government and Defense

XML is also appearing inside government agencies in both the U.S. and the rest of the world. However, the use of XML should not be surprising given that government agencies have been longtime users of SGML. Several of the U.S. initiatives are worth highlighting. As part of the eGov initiative, the XML Working Group has been doing great work in promoting XML and unified schema registries. The ability to publish both electronically and in print from the same source is important to civilian agencies seeking to move online without leaving those reliant on paper behind.

Law enforcement and military agencies are using XML to integrate intelligence data. Future versions of the Land Warrior combat system, which includes wearable computers integrated into soldiers' uniforms, will exchange real-time data with headquarters. The intelligence community already uses XML as a way to tag incoming messages and associate them with particular topics, geographical areas, and events. Incoming information is then immediately routed to the analysts who are most capable of interpreting and making use of that specific data – for example, messages about small arms fire in sector 1-D should go to Lt. Smith in analysis section 3, marked as high priority. U.S. Army Land Warrior and next-generation combat C4I systems will make extensive use of XML for communication between soldiers on the ground and in command centers (see Figure 2).

Increasingly, many counter-terrorism efforts are seen as a giant information systems integration problem. The Homeland Security Department tries to meet the requirement for a unified terrorist watch list (instead of the much-jeered 19 different lists that exist today). It struggles with fundamental data integration ills not unlike those found in the private sector, and the remedy is increasingly to use XML or Web services whenever possible.

XML Routers

The "big ideas" currently being marketed by the top IT companies are on- demand computing, utility computing, and related concepts for flexible IT cost structures. Underneath the marketing, there are some real technology requirements to implement such a vision – reducing the number of servers and applications that have to be managed by virtualizing compute, storage, and network resources. As with intelligence dispatches destined for Lt. Smith, such shared infrastructure relies on the ability to direct transactions to where they should be processed. The idea of routing transactions based on their content is not new, but XML makes it much more practical by providing a common data encoding format and standards for querying content (primarily XPath).

For example, an XML router can be used to direct all POs that are over $50K and destined for the European division to the London data center. The XML router would parse the entire XML document and evaluate an XPath expression against it, "/po[dest='EU'] and /po/details/total/usd > 50000". Application code would be designed to process any type of transaction. Changes in activity due to national holidays or natural disaster could be accommodated by simply changing the routing rule – and redirecting all POs to the U.S. datacenter, thereby shifting server resources "on demand" across the ocean.

Another application of XML routers is intelligent SOAP load balancing where a single SOAP endpoint is exposed as a front end to many SOAP servers. Emerging specifications (such as WS-Routing and WS-Referral) aim to codify ways of specifying and processing SOAP routing headers. Regardless of the application, virtualized processing resources require the ability to have transactions intelligently routed to them, a job for wire-speed XML routers.

Systems and Network Management

The use of XML in networks is not limited to XML-aware actions on application data to route, transform, or filter them. Today, network and systems management is primarily done using a menagerie of technologies: SNMP MIBs, CLI (such as Cisco's command-line interface), product-specific GUIs, syslog logs, and proprietary agent plug-ins. Of these, product-specific GUIs and CLIs tend to provide the most detailed and up-to-date information, but are the most difficult to integrate into an overall network or security management framework. (When a Fortune 500 company NOC is using screen-scraping Perl scripts to monitor and control their routers and servers, that's probably a sign that some technological shortcomings need to be corrected.) SNMP is widely supported, especially by network equipment, but

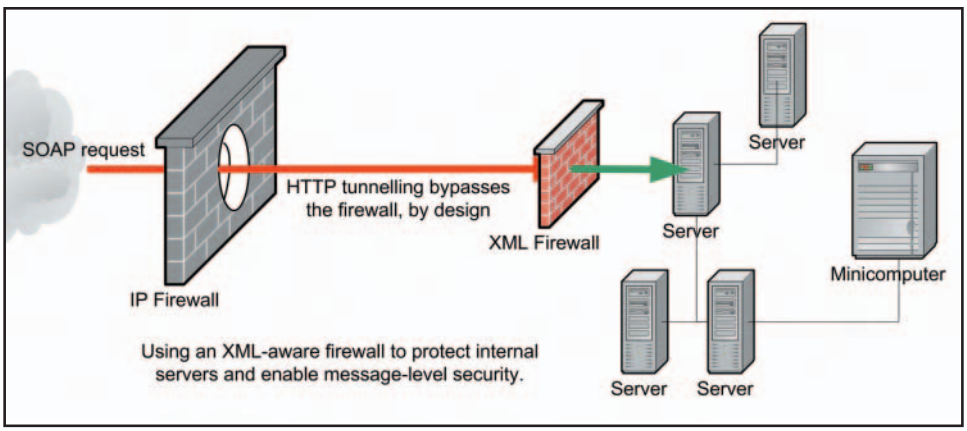


Figure 1 • Using the XML-aware firewall to protect internal servers and enable message level security

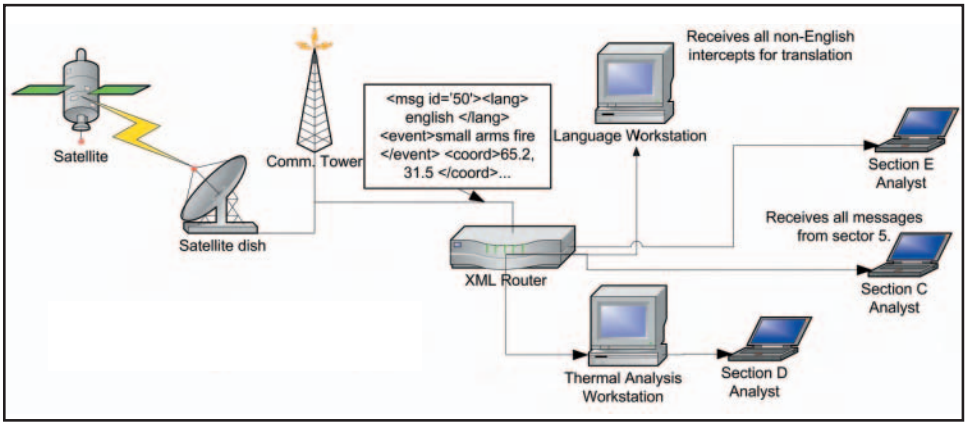


Figure 2 • XML Routing distributes information in a content aware pub-sub intelligence network for analysis

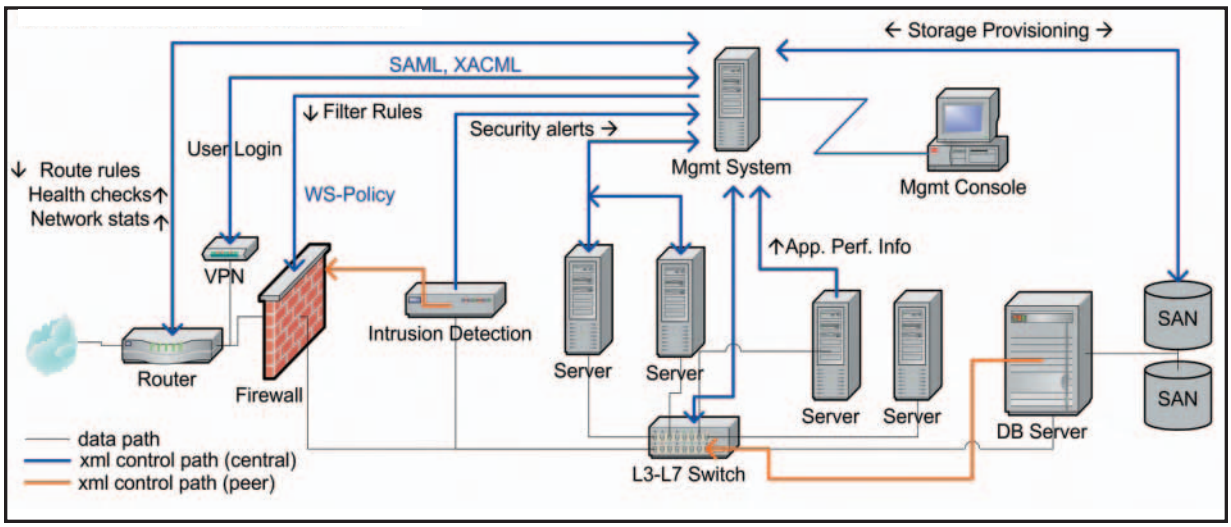


Figure 3 • XML in network and systems management

tends to expose only a fraction of a device's capability. The functionality is further curtailed by security policies that frequently limit SNMP to read-only access. Custom agents that use proprietary wire protocols to converse with the central management server usually provide more functionality than SNMP, but less than the native product UI – they are popular for app server management and single sign-on systems, and less so for network gear. Finally, logs can be captured and filtered to monitor for events of interest such as security events or response time warnings. All of this falls far short of the dream of an enterprise-wide management system, and the challenges should seem eerily familiar to readers involved with EAI and B2B integration projects.

This insight that systems, security, and network management can be viewed as an integration problem is now encouraging the use of XML as a standard way of managing networks and systems. F5 Networks has released its iControl interface, a SOAP API that enables programmatic control of their traffic management devices. DataPower offers a full WS-I BasicProfile–compliant SOAP API for our XML security gateways and XML accelerators. The three leading systems/network management vendors – Computer Associates, Hewlett Packard, and IBM – have been moving to add Web services interfaces to their products as well.

The goal is to make it easier to connect devices to management servers, enable more connections (so that every device is centrally managed), and finally to get richer information from the managed nodes. If yesterday's network management consists of a "grep" on logfiles for an alert, an SNMP call to get the number of sent packets, and a script to generate some CLI commands across a telnet session in response, today XML as the network management protocol allows all of this to happen using the same API with a better user interface. The information is also much richer – rather than the number of packets, you might be able to see the total number and value of transactions. Security policies encoded as WS-Policy or XACML are pushed out to policy enforcement nodes, which apply filters and access control to application traffic (see Figure 3).

To avoid confusion, it's important to note that, unlike XML-aware network devices such as XML firewalls or XML routers, an XML-managed device does not require that its dataplane be XML-aware – even a simple Ethernet switch can be modified to be configurable using XML. It can continue to be unaware of the XML contained in the ethernet packets it's forwarding (see Figure 4).

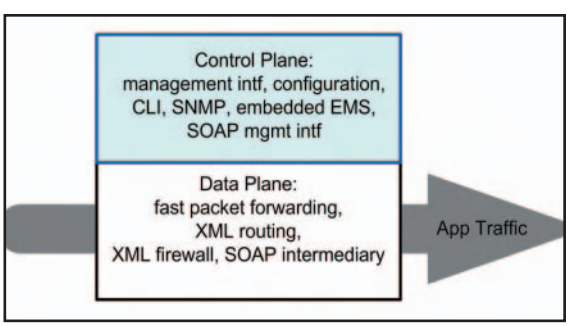


Figure 4 • Both the control plane and the data plane may be independently XML-enabled

Conclusion

Finding XML in all of these unexpected places is a great sign for adoption, but can be a little disorienting and saddening for pioneers. For them, it's not unlike the day when the rest of the world discovered the Internet, and it suddenly became something very different from what it was. Seeing the reasons for the emergence of XML outside software systems requires an understanding of the bigger picture and organizational dynamics, such as the need to control company-wide security policy outside application code. Beware of combining all XML projects into one corporate initiative – the presence of XML today is becoming so broad that such a combination may prevent all of them from making progress rather than conserve resources. After all, you wouldn't think of combining all the projects that use ASCII into one? While XML isn't quite as pervasive as ASCII yet, it's certainly starting to appear in some quite unexpected places, just like any other dominant technology.

References

- *XML.Gov:* www.xml.gov
- *CIO Council XML Working Group:* www.infoworld.com/article/03/09/01/34NNxml_1.html
- Schafer, Scott Tyler. "XML Exposes Rich Network Data." InfoWorld.
- *Jabber Technology Overview:* www.jabber.org/about/techover.html
- *Web Services Policy Framework (WS-Policy):* http://xml.coverpages.org/ws-policyV11.pdf

EUGENE@DATAPOWER.COM

International Conference & Expo

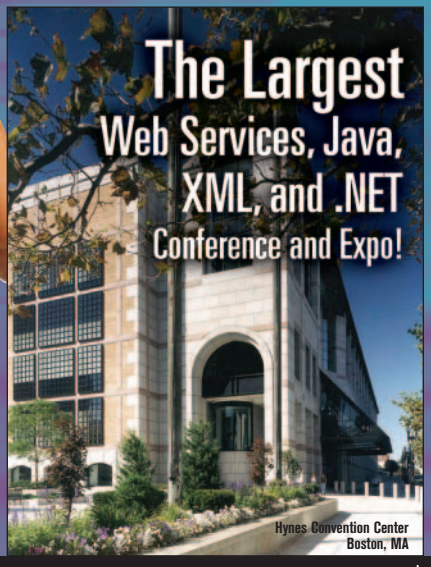
Edge 2004 EAST

Development Technologies Exchange

February 24-26, 2004

Hynes Convention Center, Boston, MA

ARCHITECTING JAVA, .NET, WEB SERVICES, OPEN SOURCE, AND XML



Addressing:

- Application Integration
- Desktop Java
- Mobility
- ASP.NET
- VS.NET
- JavaServer Faces
- SOA
- Interoperability
- Java Gaming

Register By November 21, 2003 SAVE Up To $400

For more information visit
www.sys-con.com
or call
201 802-3069

Over 200 participating companies will display and demonstrate over 500 developer products and solutions.

Over 3,000 Systems Integrators, System Architects, Developers, and Project Managers will attend the conference expo.

Over 100 of the latest sessions on training, certifications, seminars, case-studies, and panel discussions promise to deliver real world benefits, the industry pulse and proven strategies.

Full Day Tutorials Targeting

- Java • .NET • XML
- Web Services • Open Source

Conference program available online!
www.sys-con.com/edgeeast2004

Contact information: 201 802-3069 • events@sys-con.com

OWNED BY SYS-CON MEDIA

PRODUCED BY SYS-CON EVENTS

LinuxWorld Magazine · LinuxWorld.com · Java Developer's Journal · JavaWorld · WebServices Journal · Linux Business Week · XML Journal · OASIS · SD Times · eaindustry Consortium · PerfectXML · .NET Journal · asp.netPRO · WebSphere Developer's Journal · WebLogic Developer's Journal · wireless Business & Technology · CF Advisor · ColdFusion Developer's Journal · ASPstreet.com · HSPstreet.com · PowerBuilder Developer's Journal · MX developer's journal

WRITTEN BY **ENRIQUE CASTRO**

The Next-Generation Building Blocks

Reduce the amount of time and money you spend on applications

Although Intel Corporation was arguably one of the first companies in the industry to build and introduce microprocessors, the company's main business for almost 20 years after its founding was the fabrication of memory integrated circuits, not microprocessors.

The opportunity for the company to exert its most fundamental impact in the industry did not come until the late '80s: building a computer industry based on interchangeable, interoperable hardware components. This process was triggered by the confluence of two factors that turned into serendipitous opportunities in the long term.

The first was that by the mid 80s memory manufacturing became unsustainable as a viable business due to continued price pressure from Japanese manufacturers. This led to a decision to focus on the more complex microprocessor chips.

The second event was unwittingly sparked by IBM: a design team in Boca Raton, Florida, was bringing a personal computer to market. To compress the schedule, the design team decided to make heavy use of outsourced technologies, including operating system software from a then obscure company called Microsoft. The primary driver for this work was time to market. To speed up the design and integration process, IBM had to be fairly open in disclosing the specifications of the various subsystems making up a personal computer. This relative openness provided opportunities for emerging, fast-moving companies to step in and become suppliers, first to IBM, and later within the industry segments that arose.

A necessary condition for interoperability is the existence of standards that describe how the building blocks interact with each other. These standards can be de facto, either because they are adopted by an entity large enough to move the market, like IBM did with the original PC, or by default because there are no competitive alternatives. This happened with the S-100 computer bus, an early industry standard that predates the IBM PC and the Centronics printer interface. Most of the standards work today takes place through formal industry-sponsored consortia. The most prominent entity for Web services is the World Wide Web Consortium, also known as the W3C.

A computer hobbyist can go to a store today and purchase a chassis, power supply, baseboard, processor, hard drives, CD reader and writer, video card, sound card, mouse, keyboard, operating system, and applications, all from different manufacturers, and assemble a perfectly working PC. This concept of building computers from interoperable parts is so commonplace today that it won't elicit more than a passing thought in most people. Nevertheless, this concept was as alien in the '60s as building firearms out of interchangeable components was in the 18th century, or people in 1880 thinking that automobiles would be built by the thousands using identical components. This transition to mass production amplifies the economic impact of a technology by several orders of magnitude, and at the same time makes the technology available to the average consumer. The efficiencies gained bring the unit price from millions of dollars down to just a few hundred.

> "The very high-level design and loose-coupling features of Web services can lead to a radical reduction in the cost and time-to-deploy it takes to attain a certain level of functionality"

Software Building Blocks

The software industry has gone through a much longer and slower period of technology maturation. Perhaps the relative facility with which software can be changed as compared with hardware also has made it harder to ensure compatibility. Advances in the past 50 years have taken us ever closer to this goal. Building software systems out of individual applications is possible today. It is normally a heavy and expensive process applied to a fairly restricted domain. For instance, enterprise application integration (EAI) in business computing systems encompasses a set of activities – IT departments rearranging applications to support ever changing business processes. EAI costs may represent up to 50% of the life cycle cost of a software system, and several times the initial acquisition cost of the software itself. Web services technology is making it possible to build distributed applications much in the same way hardware systems are built today: by combining standards-based compatible subcomponents at a fraction of the cost of traditional EAI techniques.

How is this possible? Web services make it possible to publish a program to the Internet, and any other program can

AUTHOR BIO

Enrique Castro is a strategic technology architect and consultant who has been with Intel Corporation for 19 years. Enrique is a cofounder of Neighborhood Learning Center, a nonprofit organization providing education and tutoring services to K–12 students and senior citizens. He has MS degrees in electrical engineering and computer science and a PhD in electrical engineeering. all from Purdue University.

use or invoke the original program using standard, universally accepted Web protocols. It would be desirable for these programs to invoke each other to attain richer functionality. This is difficult to do in Web as it was originally conceived; even though it is possible to publish a program to the Web (the larger programs are known as "Web applications"), the interface is designed for humans. Although it is possible to write a program to use a Web site and "browse," it is awkward to separate the markup constructs from data. Web services technology facilitates this process by using a universally recognized data format, XML, and carrying display information separately. An XML document can be rendered into an HTML document using the XSLT standard.

A Web services–enabled program can invoke other Web services–enabled programs, which in turn call other programs and so on ad infinitum. In computer science parlance, it is said that this action, called composition, is done recursively. It's possible that even an old but still useful database running in a mainframe can be Web services enabled by adding a Web protocol layer on top, allowing the database to be used as a building block for other programs. The more traditional Web-based applications stop short of recursive composability.

Building Blocks and Quality of Service

Early content aggregation services such as Yahoo were painstakingly put together using custom programming. With Web services, similar functionality can be implemented as a program that draws content feeds from many sources, with the sources drawing content in turn from other sources, recursively. There are additional considerations for commercial services that are less relevant in a consumer setting. A paid service, say a weather information service for local farmers, needs to offer a certain quality of service, provide a service that can be metered and billed, and needs to be secure and manageable.

Web services provide an excellent foundation to attain these goals by separating data from information about the data, or metadata. This is a common design pattern used to put together complex systems. Figure 1 shows one example in hardware design: the Canon EOS system lens interface. The interface allows attaching a lens, shown on the right in the figure, to a camera body, shown at left. The most obvious function in the interface is a large opening in the camera body designed to let the data in, in the form of light collected by the lens. Notice the row of gold-plated contacts running across the bottom of the interface, both in the body and a matching set in the lens.

This is how metadata is transmitted across the interface. Metadata items include focal length, maximum and current lens aperture, distance to the object, the position of the zoom ring, and power for the focusing motor and to set the lens aperture. From the metadata, the microprocessor in the camera can make decisions on managing the camera as a system, for instance, introducing a bias to select higher shutter speeds if a telephoto lens is installed when computing exposure.

Software technology is not yet there, but as it matures, enterprise users will demand more quantitative behaviors in terms of security, manageability, quality of service, and scalability and billing. The trick here is to devise a methodology that will allow figuring out the behaviors of the composite service from the behaviors of the constituent services quantitatively.

Conceptual simplicity has been part of the appeal of Web services, with many shops experimenting with the technology in spite of its immaturity. Under the same principle of conceptual economy, it's possible to think of manageability, scalability, almost any *–ability*, as an application in its own right; meta-applications, but bona fide applications nevertheless. As applications become services, customers will have the option of hosting the manageability function in-house, or outsourcing it to a service provider. Hybrids are even possible: manageability could be hosted in-house built from subcontracted services. Subcontracting a service to build your own means renting a pluggable harness from a service provider. The service includes both data and metadata. The metadata can encompass management parameters to be integrated within the local infrastructure. XML provides an excellent medium for such exchanges.

Quantifiable manageability and QoS data coming in from subcontracters allow the same parameters going out to be published to customers downstream. Management parameters furnished by service providers can be used to build the local data center management policies. This way services are treated in a uniform way, whether in-house or subcontracted. This property would significantly reduce TCO (total cost of ownership) by radically reducing the cost of transacting and managing services, even when the constituent technology elements are pieced together from a number of equipment manufacturers and service providers.

Quantifiable QoS could create opportunities for service providers: it's possible to conceive a horizontal aggregator providing a stock quote service with a certified 99.99% uptime. A single service with that level of reliability need not exist. The aggregator can simply combine a number of independent feeds for the same data with known levels of reliability in a Web service until the desired, quantified reliability level is attained.

Web Services Technology Changes Software Design Dynamics

By design, Web services components are loosely coupled and can be separately developed. Interfaces can be changed until the moment of invocation through the self-discovery and

Figure 1 • Canon EOS system lens interface

self-configuration features of the WSDL (Web Services Description Language). In a more traditional development environment, with tightly coupled applications using DCOM or CORBA, a small change in one application can cause a chain reaction that forces fixing dozens or even hundreds of applications that depend on the initial application. This very expensive correction is usually avoided by applying strict design discipline and freezing interfaces early in the design process. The application of the design rules is still expensive, and impairs the ability of the implementers to incorporate ever-changing and fluid business requirements into the design. Some seemingly elegant architectural designs do not take into account potential nuts-and-bolts engineering side effects. Web services technology brings a capability to circumvent this problem.

It is not surprising that the very high-

level design and loose-coupling features of Web services can lead to a radical reduction in the cost and time-to-deploy it takes to attain a certain level of functionality: as much as a 90% reduction over traditional object-based methodologies with integration middleware.

The implications are much richer than just the ability to deliver a project in 10% time, although it's possible if implementation time is the main requirement. New dynamics may develop in how applications are built – it will make prototyping easier to get the functionality just right. Before, projects were so expensive that careful planning was needed to get them "right" the first time, and the notion of "right" had often changed by the time the application was rolled out.

Web services do for control what the traditional Web did for data: nodes become applications and the links are Web services APIs. The potential effect here is the blurring between the client, the front end, and the middle tier. In this environment there is much more flexibility in where the different pieces of computation can be run.

Conclusion

Like any powerful technology, Web services technology requires tradeoffs. There are performance consequences that need to be accounted for. Loosely coupled applications have inherently more latency. For instance, when compared with established design frameworks, the amount of computation per exchange under Web services goes up by at least an order of magnitude due to marshaling/unmarshaling of parameters, security processing, and conversion between binary, XML, and other formats. Network traffic goes up a similar amount because of the XML bulk as compared to binary data, and so does the communication latency because of the extra protocol layers. To give the reader an idea of the increase in data volume involved, the representation of a single note (the middle C in the piano keyboard) takes 37 lines using the MusicXML standard. In another example, a single assignment statement, inputInt = 2, encoded in 10 bytes in binary takes 11 lines and 373 bytes in an XML SOAP (Simple Object Access Protocol) invocation.

> "Web services do for control what the traditional Web did for data: nodes become applications and the links are Web services APIs"

The tradeoff is reminiscent of the tradeoff that compilers brought a few decades ago: they were initially controversial because although they saved programming time, compilers were piggish in the use of resources. This time there is no apparent controversy in this area; reducing the cost of labor takes precedence. Designers are preoccupied with addressing security and performance issues to facilitate the adoption of the technology, not in seeing these two factors as fundamental roadblocks.

ENRIQUE.G.CASTRO-LEON@INTEL.COM

WRITTEN BY **PG BARTLETT**

Seven Ways to Mess Up with XML

Avoid these common mistakes in your next project

A successful XML publishing project inspired this article. The project's leader, who claims that the financial return gained for his company "made his career" there, achieved success for two reasons: he focused on the right goals and executed the project in the right way.

This article focuses on two things: how to establish the right goals for an XML-based publishing project and the most common mistakes made. We explore the topic by discussing how to go about it the wrong way.

Mistake #1: Plan too little

Everyone knows the importance of upfront planning, right? Yet, even though "everyone knows," we regularly see projects marred by inadequate and superficial planning.

Why does this happen? Two common reasons emerge. First, most people responsible for planning grew up with word-processing and desktop-publishing software. As a result, they typically think that implementing an XML-based system primarily involves a substitution of technologies and file formats.

In reality, using XML for publishing involves new and unfamiliar concepts – it's a true paradigm shift. Unless someone with XML publishing experience helps with the planning, you will likely invest too little in the upfront work.

Second, the decision to launch an XML publishing project can take too long (doesn't it always?). But because the deadline doesn't change, planning gets squeezed to leave more time for implementing the wrong thing. Dilbert cartoons routinely illustrate this problem quite effectively.

Complicating this problem, it's also possible to go overboard on planning. This occurs much less often, but it's still costly because it delays the realization of benefits. Six to eight weeks for planning is about right. If that's not sufficient, then you're probably making mistake #2.

Mistake #2: Try to do too much at once

Once bitten by the XML publishing bug, it's easy to identify opportunities for dramatic improvement everywhere in your organization. So much waste! So much redundancy! So much inaccuracy! How could we have been so blind?

But you must resist trying to change everything at once. Too many people, too many processes, and too many document types exist to tackle everything at once. Instead, start with one group, one process, and one set of related document types.

Some words of caution: make sure you take the long view when planning so that phase VII of your project works well with phase I. You don't want every phase to require going back and changing previously completed phases.

Mistake #3: Try to change too little

Here's a surefire way to fail: start with the aim of creating "minimum disruption." Sounds good – won't work. You want to leave the same tools and processes in place and get a different result? You don't want to affect anyone or change anything but you want to achieve great benefits?

No magic beans exist. If you want to achieve dramatic results, expect to make dramatic changes. Since people naturally resist change, you will need to sell them on the organizational and individual benefits of the changes.

Mistake #4: Try to automatically convert all existing content to XML

Here's one of the most dangerous misunderstandings in publishing: existing processes and tools produce information that is sufficiently consistent to allow automatic conversion to XML. No matter how many times we have encountered that belief – and no matter how insistently it is expressed – it is always wrong.

Word-processing and desktop-publishing tools survive precisely because of the flexibility and freedom they provide to authors. These product attributes are opposed diametrically to the primary purpose of creating XML content, which involves constraining the author to create content according to a set of rules.

Is it hopeless to convert existing content to XML? Not at all. Tools are available that can convert existing content to XML. But you must accept that manual cleanup will be required, so design your process accordingly.

If you're contemplating a one-time conversion of existing information to XML, that's a subject for another article. In this article, we're focusing on building a new system that uses ongoing conversions from word processors.

In such cases, for simple documents or simple content, the manual cleanup may be minimal and, therefore, reasonable. But for long, complex documents, the cleanup cost may be excessive.

You should carefully avoid presenting a cost justification for your system that depends on ongoing, fully automatic conversion of long, complex information to XML.

AUTHOR BIO

PG Bartlett is vice president of marketing at Arbortext, where he is responsible for overall corporate positioning, branding and identity, and driving an integrated communication strategy. Bartlett joined Arbortext in 1994, bringing more than 18 years of experience in both technical and marketing positions at leading-edge high technology companies. He is a frequent presenter at major industry events and has been invited to speak and chair sessions at Comdex, Seybold Seminars, XML conferences, AIIM conferences, and others.

Mistake #5: Try to convert word-processing tools to XML editors

We have seen companies waste millions of dollars building applications on top of word processors in an attempt to force authors to conform consistently to a set of rules. Why? Because the tools do not provide the architecture that absolute conformance to a data model requires.

Fortunately, word processors and desktop-publishing software are becoming increasingly XML-aware and a few are even XML-capable. These tools offer a greater chance of success, especially if you arm yourself with expert assistance to dissect vendors' claims.

We'll explore this topic in greater detail in a future article.

Mistake #6: Set up too many rules

We're referring to the data model – the DTD or schema – that guides the author in creating and editing content. Two dimensions exist to the problems of "too many rules." First, the data model is too restrictive, and second, the data model has too many tags.

Many novices begin by designing highly restrictive data models with lots of tags. Such data models involve too many subsequent changes, which cost time and money, and require authors to spend a long time learning them.

To make a model overly restrictive, you would be very careful about limiting where tags can be used and how they can be used. For example, you may decide that a <part number> tag can appear only in a <paragraph> tag. But later you may realize that you have to allow a <title> tag to contain <part number> as well. And then you'll find still more places where you need to be able to use <part number>.

To create a problem of too many tags, give authors somewhere between 200 and 300 tags to learn so that they reach their maximum productivity just about the time that they move on to another job. If you want an overly broad generalization, shoot for 30 tags.

Mistake #7: Use too many moving parts

The problem with too many moving parts is that you must do a lot of work to choose them, integrate them, test them, and keep them all working.

In traditional publishing processes involving a lot of manual work, a problem usually doesn't erupt. Many moving parts may exist but human intervention integrates them and keeps the whole machine working. For example, contributing authors may use word processors while the technical publications department uses desktop-publishing software and manually imports the word-processor files as needed.

In an XML publishing system, however, one of the goals is to eliminate human intervention and make everything work together automatically. Fulfilling this goal requires tight integration among the various software products.

XML publishing systems must also deliver more functionality and productivity than the traditional systems they replace, so a key project requirement usually includes the execution of a content management system as well.

No single vendor offers a complete system that delivers all of the functionality needed in support of every type of content. That leaves customers with the task of selecting vendors for each piece of functionality needed.

The short answer is to limit the number of vendors involved – choose enough to accomplish your goals (both immediate and future!) but no more. The long answer is to get some expert assistance to help you match your current and future needs with the products available.

PGB@ARBORTEXT.COM

Visualizing XML in Manufacturing Systems

XML spurs SVG adoption

WRITTEN BY
ROB WILLIAMSON

The manufacturing industry has been a leader in adopting XML technologies, recognizing the benefits of enterprise-class open standards. Applications in the manufacturing industry often need to live as long as the capital equipment itself – a time frame that can stretch as long as 30 years. The need for an extended life span has driven the adoption of markup languages, guaranteeing the longevity of data and applications.

Manufacturers are turning to Scalable Vector Graphics (SVG) to extend the power of XML open standards even further. Application developers who serve the manufacturing market are mandated to update business processes to improve profitability and achieve a significant return on investment. SVG provides a powerful interface technology that makes this a reality.

This article will explore how SVG is providing significant benefits for manufacturers by Web-enabling the Human-Machine Interface (HMI). Using the same XML framework, we will discuss how these technologies can also support interactive electronic technical documentation in manufacturing. In both of these areas, traditional closed applications and publishing systems are being broken open and distributed using XML and SVG.

What Is SVG?

SVG is the W3C standard for vector graphics presentation over the Web. The standard was created by more than 20 major corporations, including AOL, Adobe, Canon, Corel, Ericsson, Microsoft, and Nokia. The most recent SVG 1.1 recommendation continues to evolve to support small devices and print. It's lightweight and can be read from any browser using a standard-compliant viewer without requiring any specialized applications running on the enterprise desktops.

SVG is XML for graphics. A human-readable markup language, SVG can be developed using specialized tools or edited in a simple word processor. As a vector format, it differs from bitmaps in that SVG is described as points, curves, and fills, and can therefore be "zoomed" into without loss of quality. Most important, because SVG is data, it can be driven by data from any location (i.e., Web services, ODCB, etc.) (see Figure 1). In addition, SVG can be made to support bidirectional interaction with the graphic, allowing the user to interact with the SVG graphic to affect changes on the target server, database, or machine.

One Technology, Many Uses

With any broad technology platform there are obvious questions. What are the best applications of this technology? What is the best way to prove its capabilities?

Some of the benefits of SVG are obvious. Like HTML and GIF, it can be applied to many applications. But SVG gives even more by replacing a baffling array of server-side graphics generation, JavaScript, Java applets, Flash applications, image servers, and Real Players with one development environment. Simply put, SVG gives all the advantages of HTML (ease of use, ubiquity, lightweight), but also enables the creation of applications that are interactive, data-driven, and rich.

As a result of these strengths, a number of opportunities are clear. SVG is an ideal presentation layer for complex information. It is an excellent choice for projects such as HMI, technical documentation, and facilities management. And because SVG is mandated to offer backwards-compatibility, it is an excellent choice for building applications that need to live for more than a few years. When existing applications are being overhauled or displaced, SVG should be a key technology employed for the next generation of systems.

In manufacturing, where complex machinery may need to live for decades, the interface to the equipment must provide the same longevity. Additionally, the solutions being implemented are built to accommodate the increasing demand and capability for customized and distributed access to disparate systems. Customers in manufacturing are relying on XML and SVG by embedding technology (i.e., SVG viewers embedded into applications) or adhering to standards. Ultimately, SVG should be deployed where multichannel communication is needed and where application longevity is key.

Let's focus on HMI as a use-case scenario and then turn our attention to technical documentation. Although they

seem different, we will explore how standards in the data and communications layers are enabling XML-based technologies to tackle both challenges in similar ways.

Industrial Automation & Human Machine Interfaces

Industrial automation means different things to different manufacturers, but from an SVG perspective, it generally includes HMI – interfacing with a machine and process monitoring. These functions share defining characteristics such as the collection, storage, and ability to look at data. And unlike some other XML workflows, they also need to access data in real time.

In fact, the report "Scalable Vector Graphics, The Future Look of HMI" by ARC Advisory Group, an industry analyst firm that covers manufacturing, had this to say about SVG and Human-Machine Interfaces:

- End users and OEMs should look to SVG as a way to display their production processes and other sources of real-time performance management information.
- End users and OEMs should look to SVG as a way to achieve some of their key business drivers, such as a reduction in the total cost of ownership due to easier installation, simpler configuration, third-party interoperability, diverse product compatibility, and scalability.
- Suppliers should evaluate their existing HMI software offerings and determine how they can incorporate SVG technology into their current or future products, delivering the benefits and key business drivers that the end users and OEMs demand. Let's take a further look at these benefits.

To understand how manufacturers can make use of SVG, we must first understand the underlying technologies. There are two standard ways to interact with a machine, through a local Programmable Logic Controller (PLC) or through networked OLE for Process Control (OPC) drivers. Although the PLC is not specifically related to the networked operation of machinery, changes implemented at the client need to be reflected in the data sent over the network so the communication is bidirectional.

Before we discuss the details of how these components fit together into a workflow with XML open standards, let's take a closer look at PLC and OPC, shown in Figure 2, which shows a graphical overview of machinery interacting with other applications in the industry, where an SVG interface is the point of human consumption of underlying corporate data from multiple repositories.

Programmable logic controllers (PLC)

PLC is a local hardware device that contains embedded software for controlling the state of a machine. For example, a PLC can be a gray-scale LCD screen with controls and intelligence that acts as a counter that shuts down a machine after a specific number of cycles. In general, PLCs are not designed to be networked, so, while the reader needs to understand their critical role in controlling machinery, they are not specifically part of the final IT project.

OPC connectivity

OPC stands for OLE for Process Control (OPC) and is used to connect devices. The standard is controlled by a consortium of equipment manufacturers. Because the majority of value is in the hardware, the vendors realized that they could sacrifice custom controls in favor of interoperability without loss of revenue related to proprietary software features. The result is that a number of low-cost and open source alternatives are available to developers looking to build applications connected to machines.

> "SVG is an ideal presentation layer for complex information"

By opening windows into the control of the devices, OPC-enabled machines can interact with other machines to enable assembly lines (or shop floors) to be combined and controlled by enterprise dashboards. This is managed via a communications layer (or driver) and represents a middleware tier to the huge variety of equipment (custom and mass market) needed in the broadly classified manufacturing vertical.

OPC servers can feed into, or play the role of, the application server tier. They optimize the communications, allow a tag-based database for multi-channel communication, and also enable the separation of data, machine, and logic layers for more rapid programming and maintenance.

It is important to note that by separating the tiers, the enterprise can integrate non-OPC systems that are critical to their operations into a common SVG interface.

SVG in Industrial Automation

SVG is a visualization layer on top of the OPC Driver layer in the enterprise infrastructure. SVG offers significant advantages in this process – SVG applications are lightweight, highly customizable, low cost, and interoperable with other XML technologies. The manufacturing floor can represent a complex

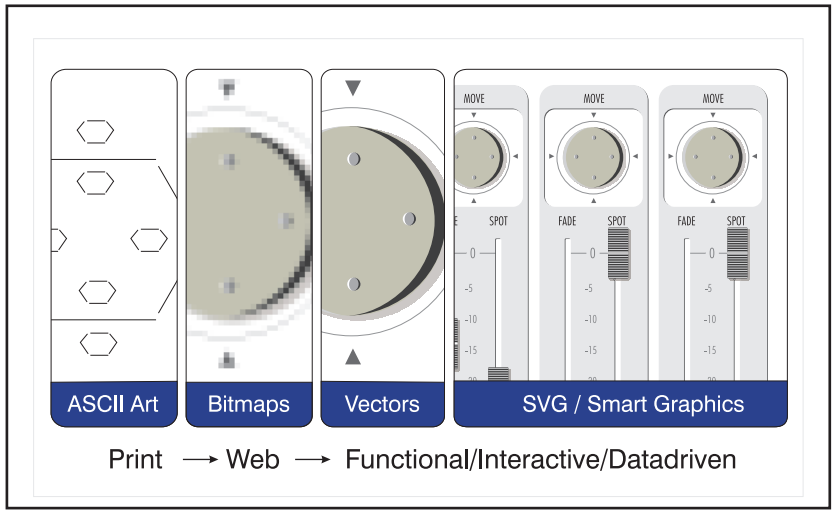


Figure 1 • The evolution of graphics

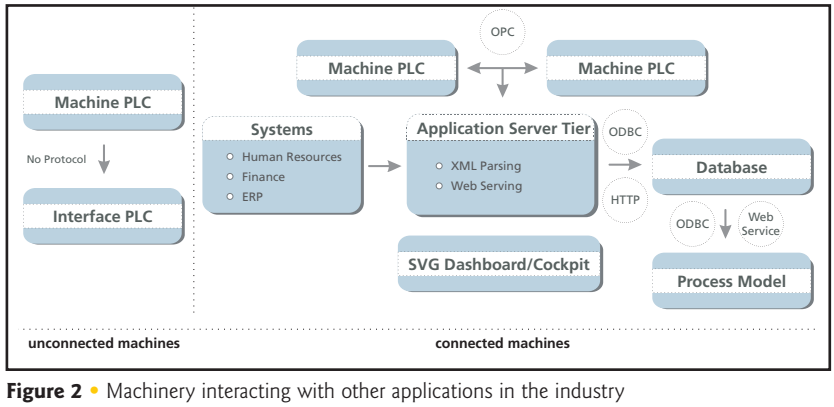


Figure 2 • Machinery interacting with other applications in the industry

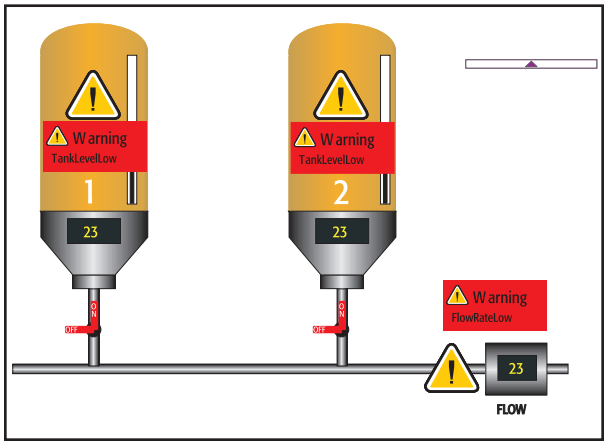


Figure 3 • Diverse types of visualization are required in complex manufacturing environments

interaction with OPC-enabled equipment, CAD floor plans, HR systems, accounting systems, and inventory management systems. Clearly, enabling the control of machines is not sufficient for realizing the dramatic improvements possible through networked control and monitoring. Combining disparate systems using XML and an SVG interface enables enterprises to integrate truly smart dashboards into their systems.

Complexity, variation, and the diverse needs of companies require the majority of effective control systems to undergo some degree of customization (see Figure 3). Customization of SVG interfaces is easy since the standard supports reuse of components, the streamlining of the design-development workflow, and a diversity of supported devices. Figure 4 shows some benefits of SVG characteristics.

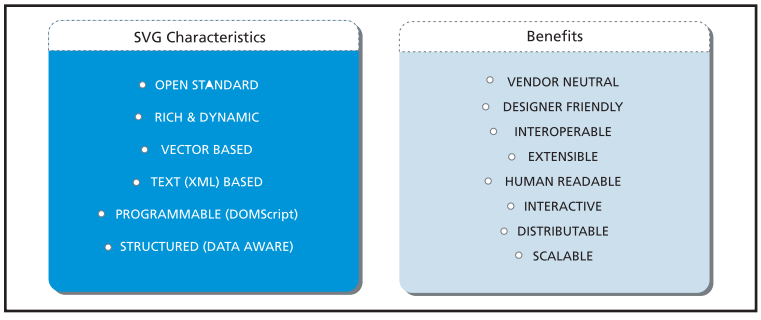


Figure 4 • Benefits of SVG

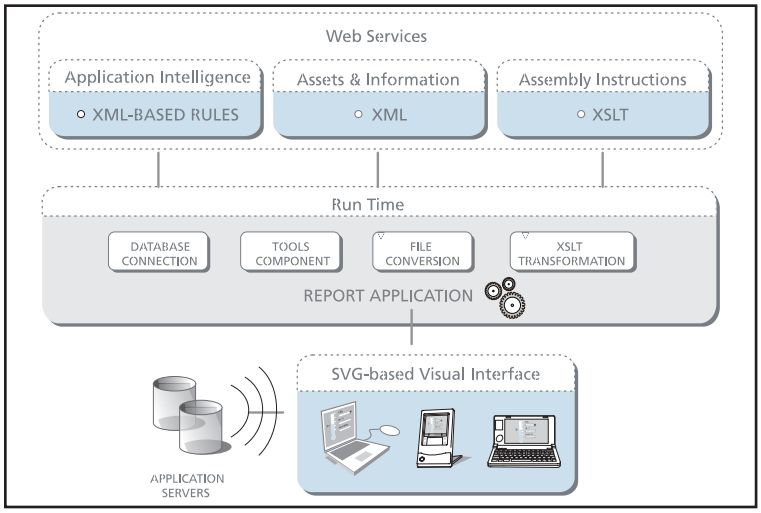


Figure 5 • Technology stack – multiple XML technologies are required to deliver dynamic interfaces from live data connections

Where Else? SVG and XML in Technical Documentation

If you are talking about SVG within the manufacturing industry, you cannot ignore the benefits provided for technical documentation. The advantages of XML for text are widely documented, but the benefits of XML-based graphics are equally compelling – particularly when you are looking to reduce the overall costs of technical documentation.

Technical documentation for capital equipment is very complex. The process involves multiple authors and technical illustrators consuming, revising, and publishing content. This content is reused in a wide variety of documents including training manuals, parts catalogues, operations manuals, and maintenance manuals. Absolute accuracy is necessary, and if the machinery is a major capital purchase, the end product is very often customized. For example, a particular model of airplane can have over 70,000 pages of documentation with content that must be customized for every airline and every aircraft based on tracking the planes by tail number.

Tech doc workflows supported by SVG

Here is a typical workflow without open standards. Technical illustrators consume CAD files using any number of commercially available vector illustration tools (like CorelDRAW or Adobe Illustrator) and export a 2D representation. The illustrator enhances the graphics, tailoring them for a specific purpose. The graphics are then passed to the technical writers who make multiple requests to the illustrators and engineers for customized versions to support their writing. And finally, if the end result is bound to inventory systems or MRO systems, a developer needs to create the associations between the graphics and the underlying data. This can be a difficult and cumbersome process.

Using SVG, the designer can use a standard, familiar tool (CorelDRAW or Adobe Illustrator) to create the graphic. The writer can use simple tools within an XML authoring platform to highlight or hotspot the portion of the graphic being described. The developer can then code the SVG to an inventory management or parts catalogue system, enabling the end users to access and update it from their PC or mobile device.

From XSLT to SVG – Tools to Get the Job Done

Previous *XML-Journal* articles have outlined how to use XSLT to generate SVG from XML (see "Database-Driven Charting Using XSLT and SVG" by Avinash Moharil & Rajesh Zade, Vol. 4, issue 5); however, this process typically applies to simple applications. When developers need to connect multiple data types from multiple sources to deliver complicated applications, they are often required to either hand code and debug a significant amount of complicated and lengthy XML language or use an IDE that can do a lot of the work for them (see Figure 5).

XSLT is an XML method to get one form of XML data into another. By leveraging Web services, XSLT, and SVG for application development, enterprises maintain a separation of the data layer, the logic layer, and the presentation layer all in XML standards.

This separation of data, logic, and presentation significantly reduces the cost of maintaining an enterprise application over its lifetime. Now, if the database administrator changes the structure of the data, the application designers and developers simply point the interface at the new data, without being required to make an overhaul to the application.

A number of products are emerging to enable the rapid creation of data-aware SVG applications. Some are closed systems, while others support easy customization. Corel Smart Graphics Studio, for example, is an attributes-based development environment that generates XSLT code in the same manner that other code generation tools (i.e., HTML editors) simplify their respective languages. Corel Smart Graphics Studio builds graphical applications that are XML from end to end. Although there are alternatives based on binary-compiled code, these technologies do not allow enterprises to leverage XML outside of the data layer and fail to take full advantage of this powerful open standard.

Interoperability Without Untenable Planning

HMI and documentation systems use almost the exact same underlying technologies to display live, interactive interfaces that combine text and graphics. They both rely on graphics file conversion XML authoring, and exchanging and interacting with data. The input models are different (OPC versus designer-generated content), but the exchange and presentation technologies are highly similar. Moreover, they also share the requirement to connect with other enterprise systems that are XML capable.

By making the investment in XML and XML-transformation technologies and adhering to open standards, enterprises can implement very large projects in a phased approach and expect interoperability. Through focusing on one system at a time, real enterprise value can be created.

The result? Unlike other IT projects that are typically plagued with a high failure rate, SVG-based projects are more successful because poor technology interoperability is no longer an issue.

Multiple Platforms, Multiple Vendors, Multiple Enterprises

SVG is becoming a widely accepted and supported standard. SVG was ratified by the 3GPPP for inclusion into wireless devices. In addition, several vendors (including Adobe and Corel) and open source communities have produced SVG viewers for PCs. Mozilla has implemented portions of the specification for the open source community and SVG has a profile available for handheld devices.

While native implementations of SVG in Web browsers are not yet available, a leading vendor recently stated that its viewer is deployed on over 70 percent of desktops. However, for the vast majority of manufacturing needs, the applications are deployed on an intranet where desktops can be easily managed by the resident IT department.

All this support means that IT projects can finally come in on time and under budget – giving CIOs a significant opportunity to show value at a limited cost. Because XML streamlines workflows, management decisions mid-stream can be easily reflected in the final application. This is in contrast to binary-compiled applications that are inflexible and cannot be easily modified after components have been implemented.

Ultimately, it is important to note that open standards are often implemented by the open source community. This puts pressure on vendors to maintain interoperability, keep pricing in line, and provide critically needed support for the SVG tools and technology that enterprises are deploying.

SVG, XSLT, XML, and the various vertical-specific standards combine nicely into powerful enterprise solutions for the manufacturing industry. With growing vendor support and powerful tools, enterprises can begin to make investments in the next generation of customizable digital dashboards to support the broad needs of their multiple stakeholders.

Author Bio

Rob Williamson, product manager, Corel Smart Graphics Studio, defines the strategic direction of Corel Smart Graphics Studio. In this role, he encourages the adoption of SVG (Scalable Vector Graphics) and works with enterprise customers to design open-standards implementations.

ROB.WILLIAMSON@COREL.COM

INDUSTRY TRENDS

~continued from page 5~

needed, depending on whether or not non-invasive integration is required.

Non-invasive multi-middleware integration requires WSDL routers that can support multiple protocols and multiple message formats simultaneously, bridging each as required. For example, if application A and application B are based on the same middleware, the WSDL router should pass their messages straight through without any conversions. For applications that use the same message formats but different transports, such as one using SOAP over HTTP and the other using SOAP over MQ, the router should perform protocol conversions but avoid unnecessary message conversions. This "on-demand" conversion approach makes the WSDL router as efficient as possible without penalizing the integrated applications with unnecessary conversions into intermediate formats or protocols.

In cases where application changes for integration are acceptable, however, multi-middleware capabilities can be added directly to each application. This allows them to talk directly to each other through their own internal multi-middleware switches, rather than relying on a centralized router. In essence, this approach is much like WSIF, except that it's not limited only to clients. It allows services to be implemented such that they too can be hosted transparently over multiple protocols and messaging formats.

While the Web services focus is often limited to only SOAP-based applications, it's clear that WSDL fulfills an even more important role in intra-enterprise consolidation and integration. Projects like Apache WSIF, and products such as IONA's Artix4, which already supports the multi-middleware routing and switching approaches described here, are using WSDL abstractions and binding extensions to maximize the value of existing IT assets. The days of expensive, invasive, and failing EAI mega-projects are over.

References

- Vinoski, Steve. "Where is Middleware?" *IEEE Internet Computing*, 6(2), Mar/Apr 2002, pp.83-85.
- *Web Services Description Language (WSDL) Version 1.2:* www.w3.org/TR/wsdl12/
- *Apache Web Services Invocation Framework:* http://ws.apache.org/wsif/
- *IONA Artix:* www.iona.com/products/artix/artix-relay.htm

Author Bio

Steve Vinoski is chief engineer of product innovation for IONA Technologies. He's been involved in middleware for 15 years. Steve is the coauthor of Advanced Corba Programming with C++ (Addison Wesley Longman, 1999), and he has helped develop middleware standards for the OMG and W3C.

VINOSKI@IONA.COM

WRITTEN BY **MICHAEL M DAVID**

Advanced ANSI SQL Native XML Integration–Part 1

A full, natural, and seamless process

This two-part article will change your view and understanding of standard SQL and its ability to integrate naturally and fully with native XML. The perceived problem with achieving full SQL-based integration of XML is that relational data is flat while XML data is hierarchical, producing a huge impediment to a seamless solution.

This belief has prevented a full integration solution, resulting in SQL vendors resorting to nonstandard SQL and external code, whose solutions fall far short of full XML integration. The usual method of integration used by SQL vendors is to shred or flatten the XML data in order to join it relationally with the relational data. This produces major efficiency problems for processing and memory utilization and ignores the hierarchical semantics in the XML data, which can be valuable.

This article demonstrates how standard ANSI SQL-92 can perform very sophisticated hierarchical processing by naturally raising the SQL level of processing to a hierarchical level. This allows SQL to integrate fully with native XML data at a full hierarchical processing level, even exceeding the nonprocedural hierarchical capabilities found in XML query languages while remaining ANSI SQL standard. This seamless XML integration is important because SQL developers require little or no training and SQL customers trust and feel comfortable with ANSI SQL.

Hierarchical Processing

A new and powerful capability that SQL hierarchical processing brings to XML is the nonprocedural hierarchical querying and processing of their multi-leg hierarchical structures. This means that the hierarchical semantics in the XML data and its structure are automatically used in the processing of the query; for example, the selecting of data in one leg of the hierarchical structure based on the value of data from another leg of the structure. Most XML query languages are based on XPath, which is primarily single path (leg) oriented and would have a difficult time correlating the semantics between two legs. But as this article will demonstrate, even this complex level of multi-leg hierarchical processing is automatically available in ANSI standard SQL syntax and semantics and can be seamlessly and directly applied to integrating native XML documents. In fact, these advanced hierarchical capabilities can perform complex XML operations such as hierarchical node promotion and fragment processing naturally and nonprocedurally in SQL.

Remarkably, the ANSI SQL capabilities that allow SQL to inherently perform full hierarchical processing come together and build on each other seamlessly, producing a truly unified and natural XML integration solution. To demonstrate this, several examples – each encompassing many aspects of this unified solution – will be used. In this way it's possible to demonstrate the big picture of SQL's complete hierarchical operation and full integration of native XML. The examples will show an unbroken progression from query start to finish of exactly how this complete and seamless XML integration is accomplished.

In Figure 1, an XML document and relational tables are joined into a larger hierarchical structure that is hierarchically processed automatically by the invoking ANSI standard SQL query. The relational tables are hierarchically modeled and joined hierarchically with the SQL modeled XML structure. The completed multi-leg structure then undergoes hierarchical data selection. The result can be returned in a relational row set whose flat result accurately reflects its hierarchically preserved data result, or it can be returned as a hierarchical XML document that, in addition to the hierarchically preserved data results, automatically reflects the query's hierarchical result structure. A hierarchical WYSIWYG display (not shown) is also possible for ad hoc interactive querying.

Hierarchical Data Modeling

Relational processing is controlled mainly by the relationships specified through the joining of tables, hence its name. Unfortunately, the standard INNER JOIN operation discards unmatched rows, creating a flat structure. With SQL-92 there is a LEFT OUTER JOIN (or just LEFT JOIN) operation that preserves unmatched data rows on the left side of the join operation. This operation is hierarchical in nature, preserving the left table over the right table because the left table rows can exist even if there are no matching right table rows, but the right table rows cannot exist without matching left table rows. The SQL-92 LEFT JOIN can string together any number of these joins to model and create a full hierarchical structure of any complexity. The LEFT JOIN has an ON clause that specifies the join criteria at each join operation. This enables the exact link points for each table (or node) in the structure to be unambiguously defined, allowing any hierarchical structure to be modeled and created. By linking to the same

AUTHOR BIO

Michael M David is founder of Advanced Data Access Technologies, Inc. He has been a staff scientist and lead XML architect for NCR/Teradata and their representative to the SQLX Group. He has researched, designed, and developed commercial query languages for heterogeneous hierarchical and relational databases for over 20 years. He has authored the book Advanced ANSI SQL Data Modeling and Structure Processing *published by Artech House Publishers, and many papers and articles on database topics.*

upper-level link point in multiple join operations, multiple hierarchical paths (legs) are modeled. This is demonstrated in the top right corner of Figure 1, where three relational tables are modeled as a multi-leg hierarchical structure using the LEFT JOIN operation.

Once a structure is modeled hierarchically by specifying its relationships as hierarchical using the LEFT JOIN, the relational engine will automatically process the structure hierarchically. The LEFT JOIN syntax enables the modeling of hierarchical structures, and the associated LEFT JOIN's semantics define the hierarchical operations to be carried out automatically by the relational engine. This will be described in more detail when we look at the relational engine and its Cartesian product processing.

Modeling relational tables is an example of modeling logical structures. This very same data modeling process can be applied to XML, which is a physical structure. In this case, the LEFT JOIN modeling must follow the physical hierarchical structure with the ON clause specifying the XML Element (or node) link points. In this way, the relational engine can process the XML hierarchical structure, properly reflecting its structure and semantics. This physical data modeling is shown in the SQL view in the upper left corner of the example in Figure 1. By performing hierarchical data modeling at the SQL view level, standard SQL has direct, full, and seamless access at the data item level for XML data values and utilization of the hierarchical semantics associated with the data.

SQL Views

Both of the data modeling examples at the top of Figure 1 demonstrate that the data modeling of different structures can be separated and stored in their own SQL view, offering significant hierarchical data abstraction, reuse, and ease of use. Since the ANSI SQL data modeling can model any kind of hierarchical structure uniformly in a standard fashion, the heterogeneous processing of different forms of hierarchical data structures becomes seamless and consistent. These hierarchical SQL views can be created by hand, or they can be created automatically by software that translates directly from data definitions such as DTDs and schemas or even COBOL.

These heterogeneous combinations of hierarchical SQL views can be combined into larger hierarchical views or structures using the same LEFT JOIN data modeling process. When joining these hierarchical views hierarchically, the left structure is modeled hierarchically above the right structure and they are linked by the ON clause join criteria, which specifies the hierarchical link point node in each structure. As shown in the double box in the middle of Figure 1, this joining of hierarchical structures can be performed by the invoking SQL statement. In fact, the invoking SQL statement joining these two previously defined structures can be specified dynamically in an ad hoc fashion. This is a very powerful and flexible combination.

You may be wondering what is combining the metadata (semantics) associated with these separate complex multi-leg views into a unified virtual view. This happens automatically when these separate hierarchical views are naturally expanded during standard SQL processing, as shown directly below the double box in Figure 1. Remarkably, the separately expanded LEFT JOIN views expand into a single contiguous LEFT JOIN specification that fully models the virtual structure being accessed as shown in the Unified SQL View diagram to the right of the Expanded SQL.

"ANSI SQL can integrate fully, naturally, and seamlessly with XML"

The dashed boxes in the Unified SQL View diagram in Figure 1 represent unselected nodes of the structure that are not output. These unselected nodes can still be utilized in the query, as node R is in one of the ON clauses. The dashed lines in the Unified SQL View represent a path of nodes that do not require access. This is because if a node is not selected for output or on a path to a selected node, then it does not require access. This optimization is possible because of the way the LEFT JOIN hierarchically preserves data, which naturally models the semantics of hierarchical data structures. As such, this optimization operates heterogeneously over the Unified SQL View composed of relational and XML data.

This powerful hierarchical data access optimization also has a significant benefit for the LEFT JOIN data modeling views. Because unnecessary paths in these hierarchical data modeling SQL views can be determined and removed at query execution, each structure needs only a single global view definition, unlike standard INNER JOIN views. This is because INNER JOIN views must always access all tables defined, regardless of the data selected, in order to keep their view consistent. This precipitates the use of multiple INNER JOIN view definitions to be created for the same structure for specific uses. The LEFT JOIN single view ability further increases the data abstraction and ease of use for hierarchical processing. This is a seamless hierarchical optimization that is driven by the selected data determined at runtime, enabling maximum ad hoc optimization.

The expanded SQL containing the two LEFT JOIN views in Figure 1 may look a little unusual because of the way the views are nested after expansion. These expanded views insert LEFT JOIN and ON clause combinations, pushing the current matching ON clause to the right. This has the nice effect of stacking the current processing and placing the current structure definition in isolation by allocating a new working set (work area) for processing the embedded view. This processing is useful, since in some cases the processing of nested SQL views can be destructive to the portion of the structure already constructed. This operation and nested syntax construction is standard ANSI SQL and is transparent. It is not usually seen or used directly by the SQL developer, because this SQL processing happens automatically under the covers.

Relational Database Engine

With the SQL fully expanded, we can look at how and why the relational engine behaves hierarchically. The simple answer is because it is carrying out the hierarchical semantics specified by the expanded LEFT JOIN specification, which is modeling the virtual and unified data structure being processed. This process naturally follows the rules and principles governing hierarchical data structures such as data nodes can have only one parent and many different children; a parent data node can exist without its children; child data nodes can not exist without their parent; and on

the more intriguing side is how sibling legs of the structure relate hierarchically when processing the query. In effect, SQL has been internally upgraded to a tree structure model (like XML) where the tables represent nodes in the structure.

Relational databases use a working set (relational row set) as working storage for processing the query. You could say that these are similar to internal temporary tables. They are flat with a fixed number of columns just like a table. Because they are fixed, it has been assumed that they could not support the processing of hierarchical data structures because hierarchical structures have multiple legs that can change in length dynamically from leg occurrence to occurrence because partial legs can exist. This happens with hierarchical structures because a parent data node can exist without its child node and so on. To represent a single data record occurrence (a root node and all of its descendent nodes) in a single row, the different legs are stored back to back. So you may assume with variable length legs that a consistent mapping of the data value locations would not be possible. But the LEFT JOIN operation cooperatively inserts null values to keep the column positions of each data value aligned. In this way, even the most complex multi-leg hierarchical structure is naturally mapped uniformly with standard relational processing. Such a Relational Working Set is shown at the bottom left side of Figure 1.

With the Relational Working Set example in Figure 1, the blank data value represents a null value that was inserted by the LEFT JOIN operation to represent missing data, preserving the structure and data. Notice how this correlates to the Hierarchical Working Data used for hierarchical engines located across from it on the opposite side of Figure 1. This same hierarchical data preserving action of the LEFT JOIN also preserves the hierarchically preserved data content correctly, otherwise the Relational Working Set row with the value "X2" would not be present.

When native XML or other hierarchical data is required, it is treated as remote data and a SQL request is sent to retrieve it. In this case, the SQL statement will be similar or identical to the LEFT JOIN view modeling the portion of the hierarchical structure being accessed. The XML document can be stored in the Internet or a column of a relational table. In our example in Figure 1, we show an XML Data Retrieval module at the bottom center that performs this operation. It retrieves the desired portion of the XML document and flattens the data into a row set (shown on its left) that represents the data defined in the requested LEFT JOIN operation (not shown). This is done dynamically so that different data requests for the same or different XML document can be processed and formatted as required for the specific query invocation. When returned to the relational engine, the row sets are joined into the relational working set at the location reserved for their hierarchical modeled portion of the unified view. This XML access is transparent to the relational engine.

Most interesting is how the relational Cartesian product engine processes complex hierarchical queries based on multiple legs; for example, selecting data from one leg of the hierarchical structure based on data from another leg of the hierarchical structure. This is the case with the invoking query example in Figure 1 inside the double box at the center of the diagram. It is selecting data from two legs of the structure using a WHERE clause to qualify (filter) the result based on data in one of the legs. It is selecting data for output based on node D values other than a "D1" value and requesting the output of the qualifying X, L, and D node values if their associated node D value qualifies. In the comparable Hierarchical Working Data diagram in the lower right side of Figure 1, you can see that the values "D2" and "D3" qualify and "D1" does not. We can also intuitively assume that the "X1" and "X2" ascendants on qualifying "D2" and "D3" path occurrences also qualify, but what about the "L1" and "L2" values on the sibling leg? Since they are both related by a common ancestor node data occurrence, "X1", they are both qualified and selected too. This is the same hierarchical WHERE processing decision logic performed by hierarchical processors.

The hierarchical query-processing logic described directly above is not trivial to carry out, so how does the relational engine perform this decision logic when it operates only on a row at a time? Either the relational engine selects a row or discards it; it does not put it aside and come back. It can do this because of the Cartesian product effect, which has produced all combinations of the data. The join relationships specified act as a data filter so that the result is a restricted Cartesian product that preserves only the meaningful and valid relationship combinations. In our case the relationships are all hierarchical. This allows the relational engine to process each row at a time and select the rows that qualify. This can be seen in the Relational Working Set at the bottom portion of the left side of Figure 1. The darkened rows are the rows that are discarded by the relational engine because they have a D1 value. Notice how both "L1" and "L2" values were selected a row at the time because of the Cartesian product effect. At this point, it has been established that hierarchical processing is a valid subset of relational processing.

Hierarchical Database Engine

With ANSI SQL hierarchical processing proven, it also establishes that the result of an ANSI SQL hierarchical syntax request is semantically correct for both relational and hierarchical processing. This can be seen at the bottom side portions of Figure 1 by comparing the Relational Working Set to the comparable Hierarchical Working Data. They have the same hierarchical data result except the row set has no way to externalize the structure, but the hierarchical structure does. This means that in the case of processing hierarchical structures, the relational engine can be seamlessly replaced by a hierarchical engine. This is shown in Figure 1 at the bottom with the box marked Relational or Hierarchical Engine. The relational engine uses a Relational Working Set shown directly to its left while a hierarchical engine uses the Hierarchical Working Data directly to its right. By using a hierarchical engine, the significant relational inefficiencies of processing and memory usage caused by the Cartesian product and its data explosions are avoided, and advanced XML hierarchical capabilities become possible. Another significant advantage occurs when the XML data is retrieved: it can be returned sdirectly as a hierarchical tree structure, which can be linked simply and very efficiently into the hierarchical structure being built as shown on the right of the XML Data Retrieval box at the bottom center of Figure 1.

Whether using a relational or hierarchical engine, the hierarchical result can be output as a Relational Result row set or as a hierarchical XML Structured Result document. This is shown

at the bottom corners in Figure 1 where the different output formats are shown. Notice that the structure of the result data structures can be different from their Relational Working Set and Hierarchical Working Data structure because the unselected nodes such as the R node are removed because they are not selected for output. This final Result Structure is diagrammed at the bottom middle of Figure 1 directly under the Relational or Hierarchical Engine box and will be used as the default XML output structure. The XML data format style (i.e. Element or Attribute centric) is controlled by the FOR XML clause in the SQL query request. It can alternatively specify an output hierarchical SQL view to non-procedurally specify specific output formats and transformations without having to introduce new procedural XML centric SQL syntax. In this example, FOR XML ELEMENT is demonstrated.

At the bottom of Figure 1, the Relational or Hierarchical Engines require knowledge of the hierarchical data structure in order to produce the XML Structured Result, perform hierarchical optimization, and for the creation and operation of the Hierarchical Working Data. This meta information is embedded in the Unified SQL View. Extracting this information is hindered because the Unified SQL View can be constructed dynamically, so it is not complete until execution. (Advanced Data Access Technologies, Inc., has developed the patented process that can dynamically determine the data structure being processed by examining the expanded Unified SQL View.)

Conclusion

Part 1 of this article has demonstrated how standard ANSI SQL can integrate fully, naturally, and seamlessly with XML by raising SQL processing naturally to a hierarchical level enabling relational data (including XML shredded data) to integrate at a hierarchical level directly with native XML. This was proven not only with examples, but by demonstrating at each stage of SQL processing how it works, from SQL syntax and semantics through the Cartesian product relational engine. It was also shown that the level of hierarchical support was significant, easily handling complex multi-legged structure queries intuitively. This allows SQL to fully utilize the hierarchical semantics in the data and the data structure. By operating at a hierarchical level the memory and processing efficiencies are greatly increased, and because SQL itself is performing the majority of the integration work, the XML support is very efficient and the footprint is very small. All of these features and capabilities support dynamic, ad hoc processing. This ad hoc processing includes powerful parameter-driven-query specification in the form of SELECT list specification and WHERE clause filtering that dynamically tailors and optimizes the most complex hierarchical query. Part 2 of this article will cover how standard SQL can seamlessly perform the more advanced XML capabilities such as node promotion, fragment processing, the handling of shared and duplicate elements, and multi-leg hierarchical data filtering. For more information see www.adatinc.com.

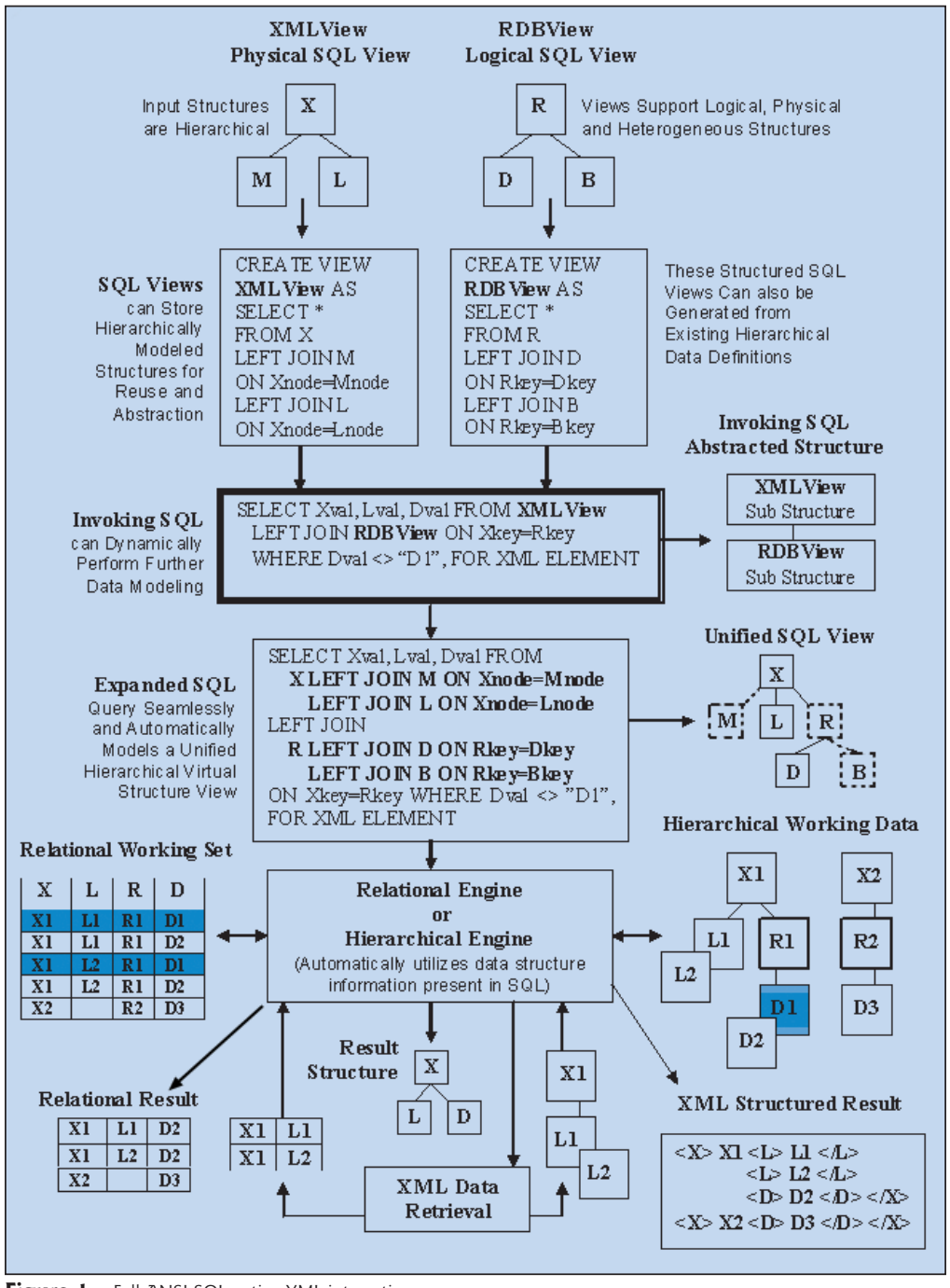


Figure 1 • Full ANSI SQL native XML integration

MIKE@ADATINC.COM

The State of Web Services, A.D. 2003

They're 'a tool for the times,' say the experts, 'and XML is key.'

What do you get if you cross an early 21st-century visionary CTO with a late 19th-century employee of the Edison Electric Light Company? Answer: a fantastic keynote address at Web Services Edge 2003 West, held in Santa Clara last month.

WRITTEN BY
JEREMY GEELAN

The visionary in question was Allan Vermeulen, coauthor of the codehead's classic *The Elements of Java Style*, and now CTO of the world's largest online retailer, Amazon.com. The Edison employee was Sam Unsell, whose contribution to the development of technology – Vermeulen explained – was to develop an economic model for electricity use in Chicago.

As with electricity then, so with Web services now. This, in Vermeulen's view, is the next shoe that needs to drop.

"Somebody has to be the Sam Unsell of Web services," he proclaimed, meaning that someone in the Web services space has to come up with a good idea for what kind of economic model is best suited to underpinning the technology.

Commercially available electricity, he explained, was only able to catch on and become pervasive because, with Unsell's help, the Edison Electric Light Company invented not just the first commercially practical incandescent lamp but a complete electrical distribution system for light and power – including generators, motors, light sockets with the Edison base, junction boxes, safety fuses, underground conductors, and other devices.

The comparison held the packed audience at the Santa Clara Convention Center, quite literally, spellbound. It was deemed by all who attended to be one of the most memorable and – pun intended – illuminating keynotes in the history of the Web Services Edge series of Conferences and Expos, which is saying something since in previous years keynotes have been given by folks like the "Father of Java," Sun's James Gosling; and the "Father of Markup," Charles F. Goldfarb.

Vermeulen's ebullient opening keynote characterized well a conference that for three days brimmed with good content and animated discussions.

The Complexity Crisis

Keynote discussion panels featured the likes of John Schmidt, CTO of the No. 1 specialty retailer in the U.S., Best Buy, who brought to bear his enormous real-world experience of Web services: Best Buy moves about 100 gigabytes of data a day – inventory data, foundation data (pricing, etc.) – and top management throughout industry, Schmidt reported, is starting to recognize the issues of complexity in IT.

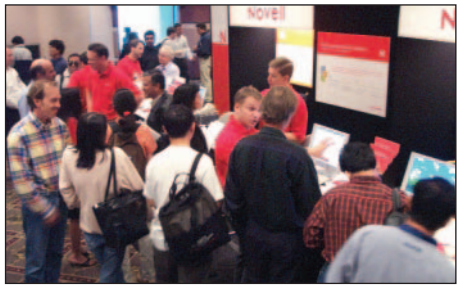
"We need," he observed, "to help take layers of complexity out of our IT environment." Whereas Web services, in Schmidt's view, may take us in the opposite direction.

Coming from a seasoned expert like Schmidt, who also chairs the Methodology Committee of the EAI Consortium, this was a compelling message – especially once he had set the stage with a reference to what he called "the dark side of systems integration – the complexity crisis."

Best Buy alone has over 600 technologies to support 165 technology capabilities, Schmidt reported. "A couple of years ago it took about 20–30 days to build a complete interface," he said. "Nowadays it takes about 4–5 days. Best Buy now adds over 550 interfaces every month (over the past 3 months)."

In other words, and this was Schmidt's point, "As complex as our environment is at the moment, Web services is going to make it even more complicated."

A Web service can be built almost at the push of a button, Schmidt concluded. "Accordingly, they will proliferate on a massive scale."



Keynote Panel: Web Services Paradigm Has Evolved

It was at another keynote discussion panel – to discuss the question "Interoperability: Is Web Services Delivering?" – that XML took center stage.

When panel moderator Derek Ferguson, editor-in-chief of *.NET Developer's Journal*, asked the panel members to set the parameters of the discussion by first defining Web services, it became clear that the invited experts on the keynote stage were agreed that, while defined by the interop protocol known as SOAP 1.1, no longer do Web services necessarily have to be XML, or even over HTTP. The paradigm has evolved.

David Chappell, VP and chief technology evangelist, Sonic Software, stressed that in his view, while Web services interactions do not have to be across HTTP, "XML is key to defining what a Web services interaction should be. It's best suited for the role of serving as the language for describing the data that needs to be exchanged between applications."

Gary Brunell, VP of professional services for Parasoft, pointed out that "If we're going to use the term 'Web services,' it does suggest the Web, and so HTTP and HTTPS. XML is very important too," he added.

Meantime, David White of Microsoft said he disagreed with the "Web" part of the term 'Web services.' "I'm a big believer in transport agnosticism," White said. "I'm really more concerned about the data representation and the invocation, rather than the transport. The key is to get something back and forth without great expense."

Chappell agreed: "To me the 'services' word is the more important, the service-oriented architecture part. 'Web services' is now a more generic term, for 'the next thing that's going to solve the problems we're trying to solve.'"

Next the panel moved on to pinpoint whether Web services has yet become common beyond the firewall, or is still mostly being used for intra-company use.

Chappell noted that in his experience there is about an 80:20 divide in terms of adoption. "80% is within the corporation's control, and 20% involves the public Internet (the Web) – dealing with other business partners, for example." Brunell agreed that mission-critical apps were still "few and far between," adding, "That's why we are all coming to these conferences."

Microsoft's White noted that on the contrary he had seen mission-critical things happen inside Web services. "We've only just gotten there," he said, "but I have absolutely seen mission-critical Web services in our customer mass." Not out in the B2B space, he conceded.

JBoss Group's CTO, Scott Stark, pinpointed one crucial piece of the jigsaw that's still missing: "Single Sign-On is a joke, I have about 35 accounts; no one has an agreement yet on a one-stop solution, and no enterprise technology can surmount that. J2EE is still basically a middleware technology," Stark continued, "it's not out there bridging enterprises."

The bridging role, then, remains perfect for Web services. But these things take time, Stark added. "Developers are going to have to get comfortable with Web services first: J2EE has taken 7 years to become a reasonably accepted technology." He pointed out that XML wasn't without its shortcomings. "XML is a double-edged sword. My head starts spinning after I've read the 10 different XML Schemas. So the usual technology curve also impedes the adoption of Web services. But that's just the nature of the beast."

Asked if XML might be replaced, White explained that one of the problems is that good tools are often the last thing to appear after a "technology burst" such as the one we are seeing around Web services. "I'm not a seer," White said, but the key to widespread adoption of any new technology is completion of the specs (we're there), demos (we're getting there), and then the tools (they're coming)."

JBoss's Stark agreed. "XML isn't going anywhere. Before there was IIOP and it went nowhere. Clearly XML is the only technology, however complex it might be, that's tried to address the problem. Besides, IIOP was even more complicated, and writing, say, a TCP/IP stack, is not a productive endeavor."

Stark then minted the phrase of the conference. "People have more comfort now with distributed programming; it's a tool for the times."

Overheard in Santa Clara

"Do you think of the 'W' in Web service as a way to ask 'Why not?' when presented with the difficulties or challenges of opening up a system or sharing information across departmental systems?"

–Velan Thillairajah

EAI Technologies, Founding Member of the EAI Industry Consortium

Binary Showdown

Do we need Binary XML?

WRITTEN BY
MICHAEL LEVENTHAL

On September 24 the W3C set in motion a process that could radically change not only how XML is used and how XML-based applications are developed – but XML itself right down to its beloved (or detested) pointy-angle brackets. "The W3C Workshop on Binary Interchange of XML Information Items Sets" brought together 34 interested parties divided among binary revolutionaries, pointy-angle-bracket fundamentalists, and a number of fence-sitters to try to form a community consensus and to decide whether or not to move forward to a full W3C activity and a binary XML Recommendation.

The price of admission to the fracas was a position paper, solicited both from within the W3C membership and from without. The positions can be reduced to four basic degrees of support or opposition to the idea the time has come for a standard Binary XML format:

1. Urgently, compellingly, immediately required. Binary XML must be standardized with or without the W3C, but preferably with.
2. Binary XML is a clear necessity, but adequate study and time must be taken to ensure a robust standard meeting a plethora of needs.
3. Not too sure about this…approach with great caution and don't undo the things that have made XML so successful. Interoperability is concern #1.
4. Not this stupid idea again! Moore's Law will solve this "problem" far faster than the W3C could.

Here is the breakdown of the how the workshop participants saw this fundamental question:

- Urgent 17
- Necessity 12
- Cautious 3
- Against 2

Workshop participants can be grouped into eight technology sectors. The positions taken showed clear segmentation across the various areas of interest as shown in Table 1.

The revolutionaries ready to storm the barricades come from the new applications for XML: wireless, digital broadcasting, and GIS. The independents (consultants, individual technologists, and none-of-the-above) showed only slightly less penchant for quick action. The more querulous old guard seems to consist of those with a greater investment in legacy XML: the technology powerhouses, database vendors, and those invovled with imaging and document applications.

I dug up the annual revenue figures for the workshop participants and calculated a revenue-weighted score for the perceived urgency of standardizing binary XML. Revenue is usually a general gauge of stake in the current technology and, as one would expect, money comes down on the side of caution. Revenue-weighted mean score: 2.376

Sometimes it doesn't matter what the industry as a whole wants to do but only what the industry gorillas want. So I looked separately at the positions of those companies with $30 billion U.S. revenue and above.

- Urgent 0
- Necessity 3 (Siemens, France Telecom, Nokia)
- Cautious 1 (IBM)
- Against 1 (Microsoft)

Why does XML need a binary format? Workshop participants identified eight major reasons:

1. ***Bandwidth:*** How many bits it takes to send an XML message across a wireless link. Wireless devices have limited bandwidth and greater use of bandwidth increases transmission time and the error rate. Binary XML can reduce the bandwidth needed to send an XML message. Compression techniques include redundancy elimination (e.g., putting common strings in a string table) or domain-based (using knowledge of the structure derived from the schema to encode content more efficiently).
2. ***Processing speed/parsing:*** Generally, XML is too slow! Binary XML addresses this problem by making the data format closer to datatypes used in programming languages and by using compression techniques to reduce the number of bits processed.
3. ***Progressive download/streaming:*** Eliminates the necessity of reading the entire XML document before it can be processed, supporting applications such as partial rendering, packetization, and interleaving.
4. ***Random access:*** Various techniques for including an index of the XML document, enabling direct access to particular sections of interest without sequential processing of the document.
5. ***Dynamic update:*** Various techniques, such as delta records, for updating an XML document without modifying the original, eliminating stream rewriting or tree node manipulations.
6. ***New data types:*** Direct support for various data types useful in non–text based applications of XML, including native numeric data types, arrays, and graphs.
7. ***Link support:*** Fundamental support for XLINK or other hyperlink primitives in core XML.
8. ***Compactness:*** Similar to limited-bandwidth approaches but

the applications usually involve archiving of very large amounts of data. En/decode time may not be as big an issue as maximal compression of the data.

How important was each of these objectives for Binary XML to the workshop participants? Given the strong turnout from the wireless world, it isn't surprising that bandwidth was the overwhelming concern (see Table 2).

Bandwidth and processing speed are not necessarily everyone's top two objectives. Eleven participants rated bandwidth as extremely important but either stated that they didn't care at all about processing speed or rated it much lower than bandwidth. Eight participants took the corresponding position in favor of processing speed and were lukewarm on the issue of bandwidth. There is a clear fracture in the interests of advocates for Binary XML. This is further illustrated by Table 3, which shows the percentage of participants within each technology domain that expressed strong or moderate preference for each objective of Binary XML.

This was not a discussion in the abstract, akin to calculating angels dancing on pins: no less than 18 existing Binary XML formats were presented. Yes, Binary XML exists today – the only question is whether the W3C will Recommend it! Of these 18 some 12 formats could qualify as "proposals"; the other 6 were described as proprietary or special purpose and not suitable for a standard. Two formats were in use by multiple participants: ASN.1 and MPEG-7 BiM. ASN.1 is a mature standard used in the telecommunications sector, which has added an XML-specific XML Schema to ASN.1 mapping. MPEG-7 BiM has also come from an established standards body, in this case in the video/television/multimedia area, which has expanded its mantle to encompass XML. A numerically oriented Binary XML, BXML from the Open GIS Consortium, was the third format which, with the previous two, was more-or-less purported to be "the answer" by its advocates. The remaining 9 Binary XML proposals were offered more in the spirit of interesting experiments that could inform the work of creating the ultimate Binary XML representation. A dichotomy in approaches to Binary XML emerged, consisting on one side of techniques that require use of the schema and on the other techniques where the Binary XML file is self-describing. Some, like Sun, felt that both techniques were needed and proposed a model which included both. Table 4 categorizes all binary formats proposed in the workshop.

A total of 14 benchmarks were presented. In the absence of a uniform methodology, data, and objectives for the format it isn't possible to say much more than that some interesting results were shown. Small XML files can be compressed with domain-specific techniques vastly better than with GZIP, no question about it. Encoding and decoding of Binary XML can be much faster than binding standard XML to programming data structures, but the numbers are fairly hazy. Everything else is very hazy if not positively chimeric. There is no credible data, for example, on the performance impact of random access or dynamic updates techniques that might be supported by a binary XML format.

One Format to Rule Them All

The W3C did a superlative job of ensuring that all opinions were heard from both the great and small and from non-W3C members as well as W3C members. On the first day papers selected by the W3C were presented by their authors to the entire assembly. All four categories of positions were given the podium, and presenters ranged from crusty individuals to representatives of the technology megaliths. On the second day the rest of the participants had a chance to present their positions to breakout groups and return to the afternoon plenary with a list of requirements for Binary XML. The final morning was spent in further discussion of the pros and cons of a Binary XML future and on various possible processes for deciding what, if anything, to do next.

"Moore's Law was invoked again and again – along with the rejoinder that mobile device batteries do not obey Moore's Law"

The tenor of the workshop was, arguably, dominated by the perspectives of the large technology vendors, with Microsoft and IBM, in particular, repeatedly cautioning participants not to throw away XML's greatest strengths – universality, interoperability, and simplicity – for the performance crisis du jour. Moore's Law was invoked again and again – along with the

Domain	# Represented	Mean Urgency
Wireless	8	1.38
Independents	8	1.88
DoD (primarily wireless)	4	1.25
Large Technology Vendor	3	2.66
Digital Broadcast	3	1.00
Database	3	2.33
Digital Imaging/SVG, etc.	3	2.00
GIS/Space	2	1.50

Table 1 • Mean urgency by domain (1=Urgent 2=Necessity 3=Cautious 4=Against)

Goal	Votes (1 for major, .5 for less important)
Bandwidth Usage Reduction	20.5
Processing Speed/Parsing	17
Progressive Download/Streaming	10
Random Access	7.5
Dynamic Update	5
Datatype Support	3
Compactness	2.5
Link Support	1

Table 2 • Goals for binary format

	Bandwidth	Parse	Stream	Random Access	Dynamic Update	Data-types	Compactness	Link
Wireless	81	63	38	25	6	0	0	0
Independents	25	63	13	38	25	0	0	13
DoD (primarily wireless)	100	13	38	0	0	0	13	0
Large Technology Vendor	50	50	0	0	0	0	0	0
Digital Broadcast	67	50	67	33	67	0	0	0
Database	67	17	33	33	0	0	0	0
Digital Imaging/ SVG	50	50	50	17	17	33	0	0
GIS/Space	50	75	0	0	0	100	100	0

Table 3 • Binary XML goals by domain (% with strong or moderate preference)

Format	Proposer	Intended Benefit	Technique
ASN.1	ISO/ITU Supporters: Sun France Telecom OSS Nokalva	Compression En/decode Speed	Schema-based compression
	Benchmarks OSS Nokalva OSS tools+PER vs. Xerces with ZIP or XMill, ZIP compression better on large files, PER 2-10X better on small files, all results better than XMill, en/decode time at least 5X better than ZIP		
	URL http://asn1.elibel.tm.fr		
BiM	ISO Supporters: NDS Expway Siemens	Compression Streaming Random Access Dynamic Update	Schema-based compression
	Benchmarks NDS Both BiM and PER ~6-20X Compression Siemens GZIP vs. BiM: similar compactness, more parsing time for GZIP due to need to decompress		
	URL http://www.mpeg-industry.com/mp7a/w4980_mp7_Overview1.html		
BXML	Open GIS Consortium Supporter: CubeWerx	Efficient Processing Dense Numeric Data	Encoding directly supports tokens, numeric datatypes, arrays, and string table
	Benchmarks CubeWerx 5-9X faster SAX-style read		
	URL http://www.opengis.org/techno/discussions/03-002r8.pdf		
CMF-B	L-3	Wire compression for ultra-low bandwidth	DTD-based compression with simplifications (ASCII, no mixed content)
	Benchmarks L-3 18:1 transmission compression		
esXML	Stephen Williams	Serialized DOM Efficient Processing Dynamic Update Random Access Streaming	Block-oriented encoding implements internal memory management, virtual pointers, and delta updates
	URL http://www.esxml.org		
FastInfoSet	Sun	En/decode Speed	XBIS-based encoding
	Benchmarks Sun Loopback request/response latency for JAX-RPC 1.6X less		
FastSchema	Sun	En/decode Speed Compression	ASN.1-based compression (schema-based)
	Benchmarks Sun Loopback request/response latency for JAX-RPC over 5X less (and better than RMI/IIOP)		
XBIS	Dennis Sosnoski	En/decode Speed	Encoding directly supports tokens, numeric datatypes, and string table
	Benchmarks 2-4X faster on SAX2 input and 4-7X faster on SAX output		
	URL http://www.sosnoski.com/opensrc/xmls/index.html		
XBVM	Lionet	Serialized "Nesting Blocks" Efficient processing, especially for multi-threaded environments Wire Size Reduction Memory Reduction	Direct compilation of document into Java classes
	Benchmarks Lionet Reduces document loading time by 2-10X, transmission size by a factor of 6, client memory by 3-4X		
Xebu	University of Helsinki Supporter: TeliaSonera	Reduced Bandwidth	Token-based compression with caching
XEUS	KDDI En/decode speed	Reduced Bandwidth	Schema-based compression
	Benchmarks KDDI 5X compression on SVG, better than XMill and ZIP. Encoding 9X faster than MXP1 microparser.		
XFSB	WebX3D Consortium Benchmarks X3D 1.6X compression	Rapid Time to Start Rendering (streaming) File Size (reduced transmission time) Reduced Parse/Load Time	Schema-based compression

Table 4 • Binary XML formats presented at the workshop

rejoinder that mobile device batteries do not obey Moore's Law.

There was universal agreement that the objectives and performance measurement criteria for Binary XML had to be formalized, and that any standard needed to have solid benchmarking behind it.

Most participants agreed that preserving XML's interoperability was of paramount importance and that the logical consequence of this imperative was that there must be no more than one Binary XML format, which will co-exist with and be easily translatable to and from standard XML. Only a few participants expressed confidence that one format would fit the bill (BiM advocate Expway and esXML proponents Stephen Williams, for two examples); others expressed open skepticism, which tended to spread quite a pall of gloom. Some were determined that the "one format" would be the one that met their needs and which were the only needs that really mattered.

And now the W3C will go off and apply scientific techniques such as hepatoscopy to workshop data and come back with a decision.

How I Read the Sheep's Liver

John Schneider of AgileDelta invoked his modified version of Metcalfe's Law to argue for enabling a new universe of XML applications with Binary XML: "the value of information exposed as XML will increase exponentially with the number of systems able to access it." Ironically, the same principle was invoked by those arguing against Binary XML. They felt that the growth of XML would slow and perhaps be permanently stunted by introducing an additional and much more complex format.

I think the numbers are with the Binary XML advocates: the wireless world is everywhere and wireless has been unequivocal about needing to standardize on a more efficient XML representation. A wireless-only standard already exists, WBXML, but the industry's current vision of the world is one where the wirelessness of a device is perfectly transparent to the network. Some of the not-wireless objectives for a Binary XML format are extremely interesting, but it does not appear that they have either the constituency or the hard data to prove their worth today. In standardizing Binary XML suitable for wireless platforms it will not be possible to trade bandwidth efficiency for features that everyone wants and thereby achieve consensus. But if there isn't one format that solves all problems there maybe some consolation for the losers. The Binary XML that emerges should give us a better core for implementing features for complex high-performance XML processing in the application layer.

Author Bio

Michael Leventhal is guiding the development of a new generation of hardware-based XML acceleration products as director, XML Technology, for Tarari (www.tarari.com). He has architected and led numerous projects in the area of Web applications and infrastructure and XML over the last 10 years, including a Web services framework and a Mozilla-based browser, DocZilla. He has spoken at numerous conferences, developed and taught the first university-level course in XML and wrote the first book on XML software development for the Internet.

Eric Lemoine, XML architect, Tarari; and Stephen Williams, senior technical director, HPTi, and founder of www.esxml.org; assisted with the research for this article.

MICHAEL.LEVENTHAL@TARARI.COM
ERIC.LEMOINE@TARARI.COM
SWILLIAMS@HPTI.COM

Using XML Schemas and DTDs Together

WRITTEN BY
GREG WATSON

DTD and schema design methods for a variety of XML development projects

XML Schemas are quickly becoming the industry standard that Document Type Definitions (DTDs) used to be. Much has been written about the advantages of XML Schemas over DTDs. Indeed, Schemas do offer advantages. However, with all the focus on the need to transition from DTDs to Schemas, it seems that little attention has been paid to how XML Schemas and DTDs can be used together.

This article focuses on how to validate an XML document against an XML Schema and a DTD at the same time. Additionally, the article focuses on how to transition from using DTDs exclusively to using both XML Schemas and DTDs. This type of transition is especially important for organizations that have heavily invested in DTDs and now have large document inventories based on them. The XML Schema and DTD in this article will be a small version of the DocBook DTD standard – a modular approach to building DTDs that has long been a standard for SGML developers. Although the example is presented in a recognizable DocBook format, it could easily be adapted to work with XML Schemas and DTDs not based on the DocBook standard. The example could also be adapted to work in a developmental environment in which a "full-version" of DocBook might be used.

A first step to consider when moving to XML Schemas is whether to write the schema from scratch, generate it from an XML document, or convert it from an existing DTD. Of these options, the first is probably the least desirable, unless development time is not an issue, which is probably not the case. A good compromise to writing a schema from scratch is to generate it automatically from an existing XML document. To generate a schema from an XML document using XMLSPY, open the XML document, select "DTD/Schema," and then select the "Generate DTD/Schema" option.

If an XML document on which a schema can be based is not available, consider generating one from an existing text file. One way to locate text-to-XML conversion programs is to search on "text to XML" on www.oreilly.com site. Unidex also offers a free trial download of XML Convert – a Java-based program for converting text files to XML (www.unidex.com/download.htm). Also, the example program ConvertToHTML in Chapter 14 of K.N. King's book *Java Programming: From the Beginning* can easily be modified to convert text files to basic XML.

In many cases though, organizations that are considering moving to XML Schemas are likely to be migrating to them from existing DTDs. The question then is how to convert these existing DTDs to XML Schemas. There are at least two known methods for converting DTDs to XML Schemas. One method is to use the Perl script DTD2Schema, which is available at www.w3.org/2000/04/schema_hack. Unlike many XML conversion utilities written in Perl, this one does not require the installation of any Perl modules. To convert a DTD to a schema using DTD2Schema, use the following command: perl DTD2Schema.pl file.dtd > file.xsd. This script will work well for converting a DTD to a schema if the DTD is contained within a single file and is not overly complex. If a DTD comprises multiple files, then converting it to a schema can best be done by using a tool such as TIBCO's TurboXML (also known as XML Authority), which is available for trial download at www.tibco/solutions/products/extensibility/turbo_xml.jsp.

To convert a DTD to a schema using TurboXML, use the following steps:

1. Open the DTD in TurboXML.
2. Select File and then select Export.
3. From the Export options, select the XSD Schema option.
4. Then provide an output file name and location for the XML Schema.

Note that after completing these steps, there is no "export complete" prompt to the user.

The converted schema will be available in the user-specified output location almost immediately after performing these steps. TurboXML works quite well in converting even the most complex DTDs to XML Schemas.

In transitioning from DTDs to Schemas, there are definitely advantages in using conversion tools such as TurboXML.

However, in using these conversion tools, be aware of two issues. First, organizations want to transition to schemas primarily because schemas allow for restricting data based on data types – integer, float, and decimal, for example. The problem in using an automated conversion tool for converting DTDs to schemas is that data types are not automatically added to the converted schema. They have to be added manually after the fact. This often goes unmentioned in discussions of automatically converting DTDs to schemas. Second, if the DTD contains text or document entities, an automated DTD-to-schema conversion tool will not convert these. The conversion tool will simply skip these entities. If they no longer need to be used in XML authoring, then this is not an issue. If there is a need to continue using them, however, two options are available. One is to include the entities in an internal DTD located in the top portion of each XML document. The other choice is to include them in an external DTD. In either case, the XML document will need to be validated against both a DTD and an XML Schema simultaneously if it contains text or document entity references.

If a task entails managing XML files with a relatively small number of entities that are used in a predictable way in XML authoring, using an internal DTD may be a good idea. Even if there are several entities to be used in document authoring, it may be advisable to use an internal DTD to define entities that may be unique to a particular XML document. To validate an XML document against an internal DTD and a schema, the following example code would need to be placed at the top of the XML file:

```
<?xml version="1.0"?>
<!DOCTYPE RootElementName [

 <!ENTITY file01.tif SYSTEM "file01.tif" NDATA tif>
 <!NOTATION tif SYSTEM "tif">
]>
<RootElementName id="RootElementID"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MySchema.xsd">
```

In this example, the internal DTD contains a single entity – a .tif image file defined as file01.tif. Any number of entities could be defined in this way within the brackets [ ] at the top of the XML file. The example also contains a reference to an XML Schema named MySchema.xsd.

Probably the most common development scenario would be a need to validate an XML document against an internal DTD, an external DTD, and a schema. Example code for doing this is as follows:

```
<?xml version="1.0"?>
<!DOCTYPE RootElementName PUBLIC "My Organization//My DTD
Description//LANGUAGE"
 "my.dtd" [

 <!ENTITY file01.tif SYSTEM "file01.tif" NDATA tif>
 <!NOTATION tif SYSTEM "tif">

]>
<RootElementName id="RootElementID"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MySchema.xsd">
```

In this example, there are really three separate validations taking place. The XML document validates against an external DTD (my.dtd), an internal DTD with an entity declaration (file01.tif), and an XML Schema (MySchema.xsd). The external DTD in this case would contain entity declarations for any text or document entities. The schema in this case would contain element and attribute declarations as well as any data type restrictions on those elements and attributes.

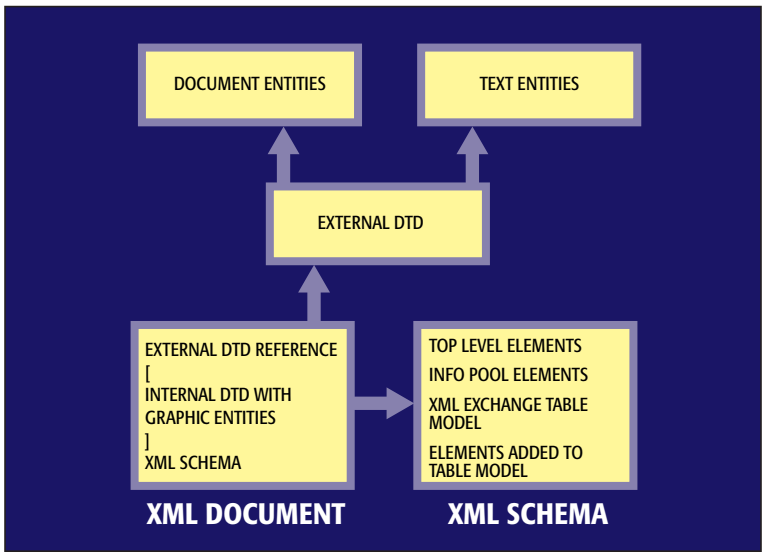


Figure 1 • Overview of the schema and DTDs

In looking at the example in the sample code (available at www.sys-con.com/xml/sourcec.cfm), consider the needs of an Internet-based company offering online access to technical white papers. In our fictional example, some of the white papers on the company site are offered free of charge, and others papers can be downloaded from the site for a fee. To better manage the data available on the site, the company has built XML files to store information about each white paper. To manage these XML files, the company has designed an XML Schema, an internal DTD, and an external DTD. Figure 1 is an overview of the schema and DTDs the company has designed.

1. The schema consists of the following components (which will be described in more detail momentarily): top-level "subject elements," DocBook "information pool" elements, the XML Exchange table model elements, and subject elements added to the XML Exchange table model.
2. The internal DTD consists of entity declarations for graphics – such as screen shots – that are unique to a particular white paper.
3. The external DTD is divided into the following parts (which will be described in more detail momentarily): DocBook ISO text entities and document entities.

In our example schema, there are four top-level subject elements – CopyrightNotice, Disclaimer, Abstract, and BiblioData. On our fictional site, the white papers offered free of charge contain a CopyrightNotice informing the reader that permission is granted to copy and distribute the paper to anyone, anywhere. The white papers on our site that can be downloaded for a fee contain a CopyrightNotice informing the reader that unauthorized distribution of the paper is punishable by a $1,000,000 fine and 10 years of hard labor in San Quinten. Each white paper contains a disclaimer protecting the company from damages resulting from loss of data or revenue as a result of using the ideas in the paper. Each white paper also

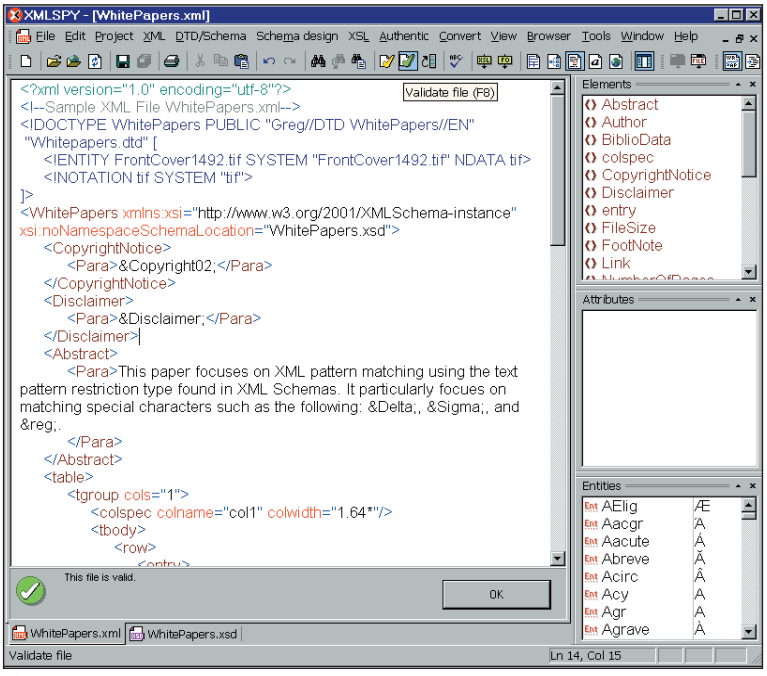


Figure 2 • Validating a sample XML document in XMLSPY

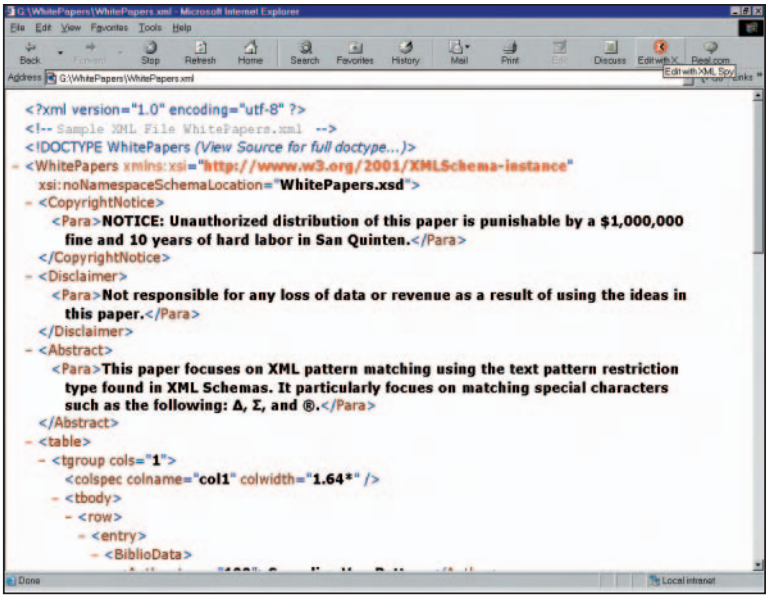


Figure 3 • XML document in Internet Explorer

contains an abstract summarizing the contents of the paper. Associated with each paper is a BiblioData element, which contains Author, Title, PaperDate, PaperNumber, PaperSubject, FileSize, and NumberOfPages elements.

The seven elements within the BiblioData element will be stored within the entry element of the XML Exchange table model – a subset of the CALS table model. By using subject elements within the entry of element of the XML Exchange model, we can more easily constrain the table data using a schema and more easily search the data using XPath queries. One additional advantage to using a CALS-based table is that the standard elements (such as entry and row) can be inserted automatically into a document using an XML editor such as Epic, which is available from Arbortext at www.arbortext.com.

To help constrain the data in the BiblioData element, we assign XML Schema data types to each of seven elements. The Author, Title, and PaperSubject elements are assigned a text data type. The FileSize element has a decimal type and the NumberOfPages element has an integer type. The PaperDate must match a year, month, day pattern (i.e., YYYYMMDD) and the PaperNumber must match the pattern of PNUM- followed by four digits.

Since many of the white papers have multiple authors, we assign a numeric value of 100 or 700 to the "type" attribute of the Author element. The value 100 identifies an author as a primary author of paper, and a value of 700 identifies an author as a coauthor of a paper. The numeric attribute values of 100 and 700 are based on the Library of Congress cataloging record known as the MARC record. For many years, the MARC record has been a standard for cataloging library materials in machine-readable format.

Recently, the MARC record has been converted to XML format. The XML Schema for the MARC record can be downloaded from the Library of Congress site at www.loc.gov/standards/marcxml/schema/MARC21slim.xsd. For more information on the MARC record in general, see http://lcweb.loc.gov/marc/umb.

In addition to the XML Exchange table model, our example schema also contains elements – such as para – that are part of the information pool module of the DocBook DTD. The information pool elements are commonly used elements in many XML document types. Our example schema will contain a handful of these commonly used elements.

The external DTD in our example contains ISO text (or character) entities and document entities. The text entities allow a document author to insert special characters or symbols into an XML document. Our example DTD contains these entities "as is" from the DocBook DTD.

Added to the DocBook entities are modules for document entities. In our example we have three document entities, which reference text file information for CopyrightNotice and Disclaimer information associated with each white paper. Declaring document entities in an external DTD allows text files to be reused in multiple XML documents as a user has a need to reference the data.

Now that we've described our XML Schema and DTDs, we can view the results in XMLSPY and Internet Explorer. Figure 2 provides a view of validating a sample XML document in XMLSPY using our example schema and DTDs. Figure 3 provides a view of this same XML document in Internet Explorer.

Summary

I've explained the ease with which you can validate an XML document against both a DTD and an XML Schema. I've also discussed how to convert a DTD to a schema and transitioning from DTDs to schemas. The DTD and schema design methods described in this article can be adapted to a variety of XML development projects. These methods provide an effective way to manage data using both DTDs and an XML Schemas.

I hope that the example in this article is helpful. Any questions about the sample code may be directed to me via e-mail; I'll be happy to help.

Author Bio

Greg Watson is a computer systems analyst working in the area of XML development at the Defense Intelligence Agency's Missile and Space Intelligence Center. In 2002, he spoke at the Intelligence Community Conference on XML Metadata.

DIWATGB@MSIC.DIA.MIL

Introducing Mapforce 2004! A powerful new XML mapping tool from Altova, producer of the industry standard XML Development Environment, XMLSPY. Mapforce is a visual data mapping tool, which auto-generates custom mapping code in multiple output languages such as XSLT, C++, C#, and Java, to enable programmatic XML-to-XML or XML-to-Database data transformations. Mapforce 2004 is the hassle-free transit for moving data from one format to another, enabling efficient information reuse. ***Download a free trial now!***

ALTOVA®

www.altova.com